Teemu Andersén

Sovelluskehyksen käyttöliittymäkerroksen uusiminen

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Koulutusohjelma
Insinöörityö
13.11.2013

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Teemu Andersén Sovelluskehyksen käyttöliittymäkerroksen uusiminen 59 sivua + 6 liitettä 13.11.2013
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Yliopettaja Erja Nikunen Järjestelmäarkkitehti Jani Lampinen

Työssä kuvataan asiakkaan suljetun sovelluskehyksen käyttöliittymäkerroksen korvaamiseksi käynnistetyn projektin vaiheet sekä lopputuotteiden tekniset ratkaisut. Projektin tavoitteena oli tuoda moderni sekä laajasti tunnettu ja tuettu käyttöliittymäteknologia/-tuote asiakkaan vanhan, suljetun sovelluskehyksen käyttöliittymäteknologian rinnalle.

Projekti oli osa asiakkaan isompaa Java-sovellusalustan päivitys -hanketta. Samaan aikaan käynnistyi toinen projekti samaisen sovelluskehyksen SOAP-kanavan uusimiseksi.

Projekti käynnistyi syksyllä 2011 vaatimusmäärittelyllä, jonka työpajoissa kerättiin järjestelmän vaatimukset sekä PoC-testauksen sisältö. Vaatimusmäärittelyn jälkeen vertailtiin eri käyttöliittymäteknologioita. Teknologiavertailun lopputuotteena oli vertailumatriisi, jota käytettiin valitsemaan käytettävä käyttöliittymäteknologia. Käytettäväksi käyttöliittymäteknologiaksi valittiin Java EE6 -standardin JSF 2.0.

Projektin toteutusosuus alkoi vuoden 2012 alussa näyttö- ja SOAP-kanavien yhteisten osien toteutuksella. Käyttöliittymäosuuden toteutus aloitettiin kunnolla keväällä 2012. Näyttökanavan toteutus muodostui kahdesta pääloppututuotteesta: käyttöliittymäarkkitehtuurista ja komponenttikirjastosta. Käyttöliittymäarkkitehtuuri rakennettiin JSF:n standardeilla sovelluslaajennoksilla. Komponenttikirjasto osoittautui isotöisemmäksi, koska se jouduttiin rakentamaan alusta alkaen itse laajennusmahdollisuuksien puutteen vuoksi.

Projekti valmistui alkuvuodesta 2013, jolloin projektin pääloppututuotteet olivat sovellusprojektien käytettävissä.

Avainsanat	JSF 2.0, CDI, Java, Web, EE5, EE6, teknologiavertailu

Metropolia

Abstract

Author(s) Title Number of Pages Date	Teemu Andersén Replacing the Web tier of a custom application framework 59 pages + 6 appendices 13 Nov 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Erja Nikunen, Principal Lecturer Jani Lampinen, Systems Architect

The customer launched a project to replace the web tier of their custom application framework. This thesis describes the steps needed to accomplish this project as well as the technical solutions used in the delivered components. The goal for this project was to bring a modern and widely used and supported web technology/product to replace the old and closed one used by the customer's custom application framework.

This project was part of a bigger initiative to update the customer's custom application platform. The first two projects started in parallel: this one was the first and the second one was to replace the SOAP channel of the same application framework.

The project was started in autumn 2011 with a requirement specification. Workshops were used to gather the requirements and the scope for proof of concept testing. After the requirement specification was finished it was time for technology comparison. The end product for technology comparison was the comparison matrix that was used to select the web technology to be applied. The selected technology was JSF 2.0, which is part of the Java EE6 specification.

The implementation phase started in the beginning of 2012 with the implementation of the common parts for the web and SOAP tiers. The implementation of the web tier started in spring 2012. The web tier implementation consisted of two separate end products: web tier architecture and component library. The web tier architecture was implemented using the JSF 2.0 with custom-built extensions. The component library was challenging, as the JSF did not provide any real extension mechanism to build custom components. Therefore, the component library had to be built from scratch.

The project was finished in the beginning of 2013 when all the end products were at the disposal of the application developers using the software platform.

Keywords	JSF 2.0, CDI, Java, Web, EE5, EE6, technology comparison

Sisällys

Lyhenteet

Ajax — Asynchronous JavaScript and XML. Tapa, jolla websovelluksiin voidaan lisätä asynkronista sisältöä.

API — Application Programming Interface. Sovellusrajapinta.

Arkkitehtuuritoimisto — Asiakkaan eri alojen arkkitehdeista koostuva yhteistyöelin, joka laatii asiakkaalle arkkitehtuurilinjauksia.

CDI — Context and Dependency Injection. Java EE -standardin versiossa 6 esitelty teknologia, joka tuo Springin kaltaisen riippuvuuksien injektoimisen ja hallinnan tavallisten Java-komponenttien käyttöön. Java EE -versiossa 5 injektiot olivat käytettävissä lähinnä EJB3- ja JSF1.2-komponenteilla.

EE5 — Java Enterprise Edition -standardin versio 5.

EE6 — Java Enterprise Edition -standardin versio 6.

EL — Java Expression Language. Ohjelmointikieli, jolla yleisemmin lisätään suoritettavia lausekkeita web-käyttöliittymäsivuille. Ohjelmointikieltä voidaan käyttää myös muualla.

Facelet — Oletusteknologia, jolla JSF-näyttöjä toteutetaan.

htp — Henkilötyöpäivä. Yhden henkilön päivän työpanos eli 7,5 h.

Java EE — Java Enterprise Edition -standardi.

JSF — Java Server Faces. Java EE -standardin mukainen web-käyttöliittymäteknologia. Java EE -versiosta 6 lähtien suositeltu tapa tuottaa Java EE -standardin mukaisia web-käyttöliittymäsovelluksia.

JSP Java Server Pages. Vanhentunut Java EE -standardin mukainen web-käyttöliittymäteknologia. Java EE 6 -versiossa korvattiin JSF:llä suositeltuna näyttöteknologiana.

Käyttöliittymästandardi
Asiakkaan ohjeistus, joka kuvaa web-sovelluksien ulkoasun ja toiminnan sekä käytettävät web-standardit jne.

MAF Meridea Application Framework, Meridealta ostettu suljettu, Java-pohjainen sovelluskehys.

MDC Meridea Design Center, Meridealta ostettu suljettu, Eclipse-pohjainen kehitin.

MDE Meridea Application Environment, Meridealta ostettu suljettu, Java-pohjainen sovelluskehitysympäristö. Koostuu MAF:sta ja MDE:sta.

OWASP Open Web Application Security Project. Avoimenlähdekoodin tietoturvaprojekti, joka ylläpitää OWASP top 10 -listaa, jossa se listaa ajankohtaiset kymmenen suurinta tietoturvauhkaa sovelluksien toteutuksissa.

PF Presentation Framework. MAF:n näyttölaiteneutraali käyttöliittymäteknologia.

PoC Proof of Concept. Teknologian/arkkitehtuurisuunnitelman konseptitason testaus, jossa validoidaan teknologian/ratkaisun soveltuvuus ratkaistavaan liiketoimintatarpeeseen.

PRG Post/Redirect/Get. Web-suunnittelumalli, jossa lomakkeen tiedot lähetetään http:n POST-operaatiolla ja näytettävä tieto haetaan uudelleenohjauksen jälkeen GET-operaatiolla.

RIA Rich Inter Application. Rikas web-sovellus, jolla on monia työasemasovellusten ominaisuuksia. Toteutetaan esimerkiksi Flashilla tai JavaScriptilla.

SiteMinder SiteMinder on keskitetty käyttövaltuushallintajärjestelmä, joka tarjoaa mm. käyttäjien autentikaation, kertakirjautumisen ja näyttöjen auktorisoinnin URL:n perusteella.

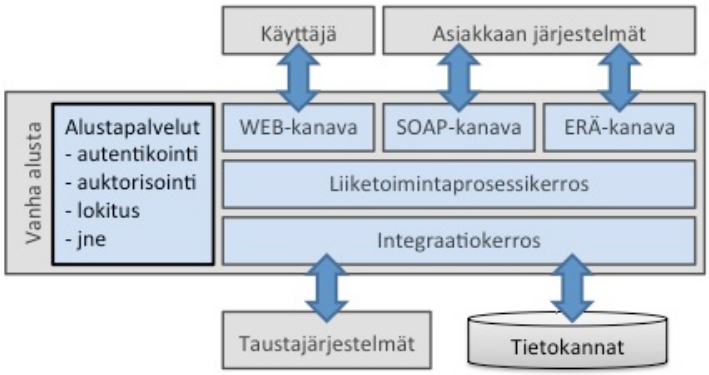
SiteMinder-agentti

http-palvelimelle asennettava agenttiohjelma, joka keskustelee SiteMinder-palvelimen kanssa ja hallitsee käyttäjien sessioita.

1 Johdanto

Asiakas on merkittävä eläkevakuutusalan palveluiden tuottaja. Asiakas ei itse ylläpidä tai kehitä järjestelmiään, vaan kaikki sovelluskehitys kilpailutetaan toimittajilla. Asiakas vastaa järjestelmätason arkkitehtuurisuunnittelusta sekä sovellustason ratkaisujen hyväksymisestä. Toimittajien vastuulla on toiminnallisuuksien toteuttaminen ja sovellusarkkitehtuurien suunnittelu. Asiakkaan ylläpidossa on useita järjestelmiä, joista Java-pohjaiset on rakennettu asiakkaan oman sovellusalustan päälle. Sovellusalusta on rakennettu alun perin Meridealta ostetun Meridea Development Environmentin (MDE) päälle, jonne on toteutettu asiakkaan toimesta laajennoksia ja yleistä toiminnallisuutta. MDE koostuu ajonaikaisesta sovelluskehyksestä Meridea Application Framework (MAF) sekä Eclipse-pohjaisesta sovelluskehittimestä Meridea Design Center (MDC). Sovellusalustaa käyttäen oli rakennettu viisi käyttöliittymäsovellusta, joista suurin sisälsi noin 270 ja muut noin 10-20 näyttöä. Meridean konkurssin myötä asiakas oli ostanut oikeudet jatkokehittää Meridean sovellusalustaa.

Meridean sovellusalustan rakentaminen aloitettiin vuonna 2002 silloisilla teknologioilla, eikä sille ole tehty juuri lainkaan jatkokehitystä tämän jälkeen. Sovelluskehys oli rakennettu aikana, jolloin Java EE -standardit eivät mahdollistaneet nykyisen kaltaista tehokasta työskentelyä, vastaavia kehyksiä ja kilpailevia teknologioita kehitettiin yleisesti. Tämän vuoksi sovelluskehyksen kaikki osat ovat erikseen rakennettuja eivätkä pohjaudu yleisiin standardeihin. Asiakkaan vanhan sovellusalustan rakenne on kuvattu kuvassa 1.



Kuva 1. Vanhan sovellusalustan rakenne

Syksyn 2010 aikana käynnistettiin sovellusalustan kehittämiskohteiden etsiminen. Haastatteluissa alustan käyttöliittymäkerros (WEB-kanava) nousi selkeästi esille hankalimpana ja aikaa vievimpänä osana sovelluskehitystä, jonka vuoksi tämä valittiin ensimmäisenä jatkokehitykseen. Lisäksi jatkokehitykseen valittiin SOAP-kanava lähinnä vanhan ratkaisun teknisten rajoitteiden vuoksi. Näille molemmille uudistuksille käynnistettiin omat rinnakkaiset projektinsa.

Tämän työn tavoitteena on kuvata käyttöliittymäkerroksen uusimisen vaiheet sekä projektin lopputuloksina syntyneen käyttöliittymäarkkitehtuurin ja komponenttikirjaston toiminnallisuus ja tekniset ratkaisut.

Riskit

Mikäli käyttöliittymäkerrosta ei uudisteta, riskinä on, että asiakkaan sovelluskehyksen käyttöliittymäkerros ”lahoaa” käsiin. Tästä voi seurata tilanne, jossa joudutaan turvautumaan mittaviin uudistustöihin pienellä aikataululla esimerkiksi sovelluspalvelimen tasonnoston yhteydessä. Tällöin kaikki vanhaa sovellusalustaa käyttävät, käyttöliittymiä sisältävät sovellukset jouduttaisiin päivittämään samalla aikataululla.

Sovelluskehyksen käyttöliittymäteknologia, Presentation Framework (PF) ja sitä tukevat työkalut ovat pääosin suljettuja, joten sovelluskehittäjiltä harvemmin löytyy aikaisempaa kokemusta ko. teknologioiden käytöstä. Siten oppimiskynnys on suuri. Lisäksi suljettu teknologia vaatii jatkuvaa ylläpitoa, kun sen toteutuksessa käytetyt teknologiat vanhenevat, ja niiden tuki esim. uusille Java-versioille päättyy.

Toteutustapa

Projekti toteutettiin käyttäen iteratiivista kehitystapaa. Kehityssyklien pituus oli kaksi viikkoa, jonka jälkeen demonstroitiin syklin aikana saavutetut tulokset ja priorisoitiin tehtävät seuraavalle syklille.

Uusi arkkitehtuuri rakennetaan siten, että sitä voidaan käyttää vanhan arkkitehtuurin kanssa rinnakkain.

Taulukossa 1 on kuvattu projektiryhmä vastuualueineen.

Taulukko 1. Projektiryhmä

Henkilö	Rooli ja vastuualueet
Asiakkaan sovellusalustan omistaja	Asiakkaan edustaja, tuoteomistaja (PO) projektin alkaessa, linjavetojen ja ratkaisujen hyväksyntä.
Sovellusalustan järjestelmäarkkitehti	Asiakkaan edustaja, sovellusalustan omistajan tekninen tukihenkilö, tuoteomistaja (PO) projektin loppuvaiheessa, teknisten linjavetojen ja ominaisuuksien hyväksyntä. Toteutuksen priorisointi.
Insinöörityön tekijä	Vaatimusmäärittelyn kerääminen ja tuottaminen, näyttöteknologian arkkitehti ja pääkehittäjä, teknisten linjausten tuottaminen, projektin aikana Scrum-master sekä projektin työlistan tuottaja ja ylläpitäjä tuoteomistajan puolesta.
Kehittäjä (x2)	Ominaisuuksien toteuttaminen

Projektin vaiheet ja alustava aikataulutus on kuvattu taulukossa 2.

Taulukko 2. Projektin vaiheet ja alustava aikataulutus

Aikataulu	Kokonaisuus
2011/Q3	**Vaatimusmäärittely ja projektisuunnittelu** - tehdään tarvittavat projektin ohjaukseen liittyvät suunnitelmat sekä selvitetään uudelle näyttöteknologialle esitetyt ja mahdollisesti tulevaisuudessa esitettävät vaatimukset.
2011/Q4	**Toteutustekniikan valinta** - selvitetään mahdolliset toteutusvaihtoehdot, tutustutaan paremmin muutamaan vaihtoehtoon, ehdotetaan sekä asiakkaan johdolla päätetään toteutuksessa käytettävä näyttöteknologia. Laaditaan arkkitehtuuriratkaisun alustava suunnitelma valitun teknologian mukaisesti.
2012/Q1	**Perustoiminnallisuuden toteuttaminen** - rakennetaan näyttö- ja SOAP-kanavien yhteiset osat, näyttökanavan perustoiminnot, näyttöjen mallipohjat ja peruskomponentit sekä SiteMinder-integraatio. Viimeistellään näyttökanavan arkkitehtuurikuvausdokumentti.
2012/Q2	**Jatkokehitys ja pilottitoteutus** - rakennetaan yleisimmin käytetyt käyttöliittymäkomponentit sekä kohdetoimijanvaihtamiseen liittyvä toiminnallisuus. Jatkokehitetään alustaintegraatiota liiketoimintaprosessien ja vanhalla näyttöteknologialla toteutettujen näyttöprosessien osalta. Pilotoidaan näyttöteknologiaa konvertoimalla ensimmäinen sovellus. Selvitetään, mitä muutoksia tarvitaan koostamisskripteihin ja asiakkaan testaustyökaluun.

Aikataulu	Kokonaisuus
2012/Q3	**Toteutuksen viimeistely ja työkalumuutokset** - viimeistellään näyttökanavan toteutus ja konvertoidaan seuraavat sovellukset sekä niiden testitapaukset, suoritetaan integraatio- ja suorituskykytestaaminen, viimeistellään integraatio Tukipalveluiden koostamisskripteihin ja toteutetaan tarvittavat MDC-muutokset.
2012/Q4	**Käyttöönottoon liittyvät tehtävät** - suunnitellaan näyttökanavan tuotantokäytön aloittamiseen liittyvät asiat, viimeistellään dokumentaatio sekä pidetään tarvittaessa ensimmäiset näyttökanavan koulutustilaisuudet.
2013 →	**Ylläpitovaihe** - uusi näyttöteknologia käytössä uusia järjestelmiä varten. Olemassa olevien järjestelmien siirtäminen uudelle teknologialle oman aikataulun mukaan. Vanhaa toteutusta tuetaan myöhemmin määriteltävän siirtymäkauden ajan.

2 Uudistuksen tavoitteet

Uudistuksen tavoitteena on tuoda moderni sekä laajasti tunnettu ja tuettu käyttöliittymäteknologia/tuote vanhan teknologian rinnalle. Tällöin teknologia on toimittajille entuudestaan tuttua ja siitä löytyy ulkopuolista dokumentaatiota, oppaita ja koulutusta. Tunnetut teknologiat parantavat toimittajien toimitusvalmiutta, vähentävät työmääriä ja lisäävät asiakkaan houkuttelevuutta sovelluskehittäjille.

Käyttöliittymäkerroksen toteutus on asiakkaan sovellusalustan laajimpia ja kompleksisimpia osakokonaisuuksia. Korvaamalla käyttöliittymätoteutus standardilla teknologialla saadaan asiakkaan ylläpitovastuulla olevia komponentteja vähennettyä kerta heitolla. Tämä vähentää merkittävästi asiakkaan ylläpitokustannuksia pitkällä aikavälillä.

Kun käytetään standardeja teknologioita, saavutetaan teknologialle pitkä elinkaari lukittumatta kuitenkaan tiettyyn toimittajaan. Lisäksi esimerkiksi Java EE -standardin mukaiset teknologiat päivittyvät sovelluspalvelimen mukana. Standardeille teknologioille löytyy laajasti osaamista ja tukea - myös kaupallista.

Käyttöliittymäkerroksen uusimisen yhteydessä voidaan lisätä tukea myös uusille käyttökokemusta parantaville teknologioille, kuten esim. AJAXille, jolloin loppukäyttäjille saadaan tarjottua käyttäjäystävällisempiä ja käytettävämpiä näyttöjä.

Arkkitehtuuritoimisto esitti oman visionsa uudistetusta käyttöliittymäkerroksesta, jonka sisältö oli seuraavanlainen:

- Järjestelmien käyttöliittymät tarjoavat käyttäjille visuaaliselta ilmeeltään selkeitä ja yksinkertaisen tyylikkäitä näyttöjä, jotka toimivat nopeasti ja virheettömästi auttaen käyttäjiä tekemään oman työnsä ripeästi ja hyvillä mielin.
- Näyttöjen teknologiaratkaisut tukeutuvat yleisesti käytössä oleviin standardeihin ja todennetusti toimiviin ratkaisuihin. Ratkaisut ovat myös teknisesti selkeitä ja yksikertaisia.
- Näyttöjen tekninen toteuttaminen, ylläpitäminen ja testaaminen on helppoa ja onnistuu vaivattomasti myös rajallisesti teknistä kokemusta omaavilta ihmisiltä.
- Pyritään pienentämään ylläpidettävän alusta-koodin määrää.
- Korvataan valitut alustan toiminnallisuudet standardien mukaisilla valmiilla toteutuksilla.
- Tehdään arkkitehtuurimuutokset, joilla varmistetaan alustan elinvoimaisuus.
- Tehostetaan sovelluskehitystä asiakkaan omalla sovellusalustalla.

Rajaukset

Tarkoituksena ei ole tuottaa näyttölaiteneutraalia käyttöliittymäteknologiaa. Tällöin voidaan panostaa selainkäyttöliittymäpuoleen ja sen standardeihin.

Projektin puitteissa ei tuoteta uusia työkaluja, vaan käytetään myöhemmin valittavaa teknologiaa varten kehitettyjä kolmannen osapuolen valmiita työkaluja soveltuvin osin.

Projektin puitteissa ei myöskään tuoteta konversiotyökalua vanhojen näyttöjen ja käyttöliittymäprosessien konvertoimiseksi uudelle teknologialle. Vanhat käyttöliittymät ja niihin liittyvät komponentit eivät ole määrämuotoisia, joten konversiotyövälineessä tulisi varautua lukuisiin poikkeuksiin. Lisäksi käyttöliittymäprosessit ovat erittäin monimutkaisia ja täynnä lukuisia poikkeuksia ja poikkeuksen poikkeuksia. Näiden seikkojen vuoksi konversiotyökalu tukisi hyvin suurella todennäköisyydellä vain pientä osaa näyttöjä ja niitäkin vain osittain. Tällöin konversiotyökaluun panostettu aika olisi paremmin käytetty itse näyttöjen uudelleentoteutuksessa.

3 Vaatimusmäärittely

Koska projekti toteutettaisiin käyttäen iteratiivista kehitysmallia, oli vaatimusmäärittelyn tarkoituksena tuottaa pohjatilanne projektin tehtäville. Vaatimusmäärittelyn sisällön oli tarkoitus olla luonteeltaan alustava listaus rakennettavan järjestelmän ominaisuuksista, jotta saataisiin aikaiseksi yleiskuva järjestelmästä ja tunnistettaisiin eri ominaisuuksien keskinäiset riippuvuudet sekä riippuvuudet muihin projekteihin. Listan pohjalta saataisiin laadittua alustava priorisointi ja aikataulutus ominaisuuksille.

Vaatimusmäärittelyn aluksi mietittiin, mistä saataisiin kerättyä vaatimusten esiehdot. Esiehtojen lähteiksi tunnistettiin seuraavat

- asiakkaan olemassa olevien sovellusten näyttöjen erityispiirteet
- asiakkaan uusi käyttöliittymästandardi, jota tuotettiin vaatimusmäärittelyn aikana
- asiakkaan arkkitehtuuripäätökset
- nykyisen web-ratkaisun ongelmakohdat.

Tämän lisäksi esitettiin toiveet, että

- valittaisiin laajasti käytetty ja elinvoimainen ratkaisu, joka olisi myös mielellään avointa lähdekoodia
- valittavalta teknologialta vaaditaan uskottavuutta pitkäaikaisten järjestelmien rakentamiseen
- uutta ja vanhaa käyttöliittymäkerrosta voitaisiin käyttää järjestelmässä rinnakkain, jolloin uusi teknologia voitaisiin ottaa käyttöön vaiheittain.

Esiehdot kerättiin haastattelemalla olemassa olevien sovellusten omistajia ja ylläpitäjiä sekä käymällä läpi tunnistettua aineistoa. Loput vaatimukset päätettiin kerätä sidosryhmiltä työpajojen avulla. Työpajatyöskentelyä varten tunnistettiin määrittelyssä tarvittavat sidosryhmät ja sovittiin agendat työpajoille. Työpajojen järjestämisen jälkeen varattiin aikaa vaatimusten kokoamiselle. Tässä yhteydessä laadittiin myös esitys PoC-testauksen laajuudesta.

Koottu vaatimusmäärittely katselmoitiin sidosryhmien ja projektin ohjausryhmän toimesta. Määrittely täydennettiin katselmointikommenteilla, jonka jälkeen se hyväksyttiin projektin ohjausryhmän toimesta.

3.1 Sidosryhmien tunnistaminen ja työpajatyöskentely

Sidosryhmiksi tunnistettiin arkkitehtuuritoimisto, tekniseltä puolelta sovelluspalvelintuki-, paketointi- ja suorituskykytestaustiimit sekä liiketoiminnalliselta puolelta isoimman sovelluksen ja pienen, mutta monimutkaisen sovelluksen vastuuhenkilöt. Vaatimusmäärittelyn työpajat järjestettiin sidosryhmien kanssa seuraavalla jaottelulla.

Työpaja1: Päälinjaukset. Työpajan tavoitteena oli vaatimusmäärittelyn pohjustus ja näkökulman valitseminen.

Työpaja2: Liiketoiminnalliset vaatimukset. Työpajan tavoitteena oli liiketoiminnallisten vaatimusten kerääminen ja läpikäynti aiemmin kerätyn määrittelyrungon pohjalta.

Työpaja3. Työpajan tavoitteena oli varmistaa, että olemassa olevien sovellusten erilaiset liiketoiminnalliset vaatimukset on huomioitu määrittelyssä.

Työpaja4: Tekniset vaatimukset. Työpajan tavoitteena oli kerätä tekniset vaatimukset, kuten tietoturva, tuotanto- ja testiympäristöt sekä standardit.

3.2 Vaatimusmäärittelyn sisältö

Työpajojen järjestämisen jälkeen varattiin aikaa vaatimusten kokoamiseen. Vaatimukset koottiin esiehtojen ja työpajojen kommenttien ja linjausten perusteella.

Vaatimusten kirjoittamisessa käytettiin IETF:n (Internet Engineering Task Force) RFC 2119 -mukaista terminologiaa [1] vaatimusten luokitteluksi. Vaatimuksille kirjoitettiin kuvaus-, seuraukset-, testaus- ja lähde-kappaleet. Kuvaus sisältää vaatimuksen kuvauksen sekä RFC 2119 -mukaisen pakollisuusluokittelun. Seuraukset kuvaavat, mitä hyötyä tai rajauksia vaatimuksesta seuraa. Testauskappaleessa kuvattiin vaatimuksen testausvaatimukset ja mahdolliset testauksen erityispiirteet. Lähdekappaleeseen lueteltiin, mistä lähteistä vaatimus on noussut, esim. työpaja 1 ja arkkitehtuuritoimisto. Liitteen 1 *Rajaukset* rajaukset on kirjoitettu tällä notaatiolla.

Vaatimusmäärittelyn sisältöä ei lähdetä avaamaan tässä yhteydessä toiminnallisten ja ei-toiminnallisten vaatimusten osalta. Rajaukset liittyvät osittain teknologian valintaan,

siksi ne on esitelty lyhyesti. Vaatimuksia kerättiin yhteensä 157 kappaletta ja rajauksia 12 kappaletta. Taulukossa 3 on esitetty lyhyt yhteenveto vaatimusmäärittelyn lopputuotoksista toiminnallisten ja ei-toiminnallisten vaatimusten osalta statistiikan muodossa.

Taulukko 3. Yhteenveto toiminnallisista vaatimuksista

Kategoria	Aihealue	Vaatimusten lkm
Toiminnalliset	Sivupohjat ja yleiset komponentit	60
Toiminnalliset	Työkalut	11
Toiminnalliset	Sekalaiset	8
Toiminnalliset yhteensä		79
Ei-toiminnalliset	Käytettävyys	14
Ei-toiminnalliset	Toimintavarmuus	6
Ei-toiminnalliset	Suorituskyky	4
Ei-toiminnalliset	Ylläpidettävyys	13
Ei-toiminnalliset	Arkkitehtuuri	16
Ei-toiminnalliset	Tietoturva	19
Ei-toiminnalliset	Testaus	6
Ei-toiminnalliset yhteensä		78
Kaikki yhteensä		**157**

Vaikka kaikki rajaukset eivät tarkkaan ottaen olekaan suoranaisia rajauksia, on ne esitetty tässä sellaisina alkuperäisen vaatimusmäärittelymateriaalin mukaisesti. Rajaukset löytyvät kokonaisuudessaan liitteestä 1. *Rajaukset*.

Rajausten kuvaukset:

- Järjestelmä täytyy toteuttaa Java-ohjelmointikielellä.
- Kaikkien järjestelmän rakentamiseen käytettävien komponenttien, kirjastojen ja työkalujen lisenssien täytyy olla asiakkaan lisenssikäytäntöjen mukaisia. Kaik-

kein järjestelmän rakentamiseen käytettävien komponenttien ja kirjastojen täytyy olla asiakkaan erikseen hyväksymiä.

- Järjestelmän, mukaan lukien sen alikomponentit, täytyy tukea kaikkia asiakkaan tukemia selaimia. Vaatimusmäärittelyä laadittaessa näitä olivat: Firefox 6, IE 7, IE 8 ja IE 9.
- Kaikkien järjestelmän rakentamiseen käytettävien komponenttien täytyy olla yhteensopivia asiakkaan sovelluspalvelimien kanssa.
- Järjestelmän olisi hyvä pohjautua avoimenlähdekoodin ratkaisuihin. Kaupallinen tuki ei ole välttämättömyys, mutta toivottavaa.
- Järjestelmän täytyy käyttää käyttöliittymästandardin määrittelemiä web-standardeja. Näitä ovat:
    - XHTML 1.0 Strict (http://www.w3.org/TR/xhtml1/)
    - CSS 2.1 (http://www.w3.org/TR/CSS/)
- HTML5 ja CSS3 -teknologioita ei saa käyttää ennen kuin ne ovat laajasti käytettyjä ja lisätty käyttöliittymästandardiin.
- Järjestelmän täytyy käyttää käyttöliittymästandardin määrittelemää JavaScript-versiota. Käytettävä versio on JavaScript 1.5 (ECMA-262 Edition 3).
  Asiakas on saanut palautetta omien asiakkaidensa tietoturvavastaavilta liittyen JavaScriptin käyttöön. He näkevät JavaScriptin mahdollisena tietoturvauhkana ja toivoisivat, että järjestelmää olisi mahdollista käyttää ilman JavaScriptiä.
- Järjestelmän täytyy käyttää käyttöliittymästandardin mukaisesti UTF-8-merkistöä.
- Järjestelmä täytyy rakentaa käyttäen asiakkaan koodausstandardeja ja muotoiluja.
- Järjestelmän täytyy käyttää käyttöliittymästandardin mukaisia kuvaformaatteja, joita ovat PNG, JPEG ja GIF.
- Järjestelmän ei täydy tukea muita laitteita kuin selaimia. Mikäli tarve uudelle laitteelle nousee, voidaan sille lisätä tuki uutena kanavana.
- Järjestelmän täytyy olla sovelluspalvelinneutraali. PoC- ja pilottitestaus tehdään käyttäen Apache Tomcat -sovelluspalvelimia. Systeemitestivaiheesta eteenpäin käytetään IBM:n WebSphere sovelluspalvelimia.

3.3 Teknologiavertailun testauksen sisältö ja laajuus

Teknologiavertailussa valituille teknologioille suoritetaan lopuksi PoC-testaus ennen lopullisen teknologian valintaa. PoC-testaus tarkoittaa testisovellusten tuottamista eri teknologioilla, joilla testataan teknologian ja sen toimintatapojen käyttökelpoisuutta määriteltyyn tilanteeseen. PoC-testauksen sisältö ja laajuus tulee määritellä ennen varsinaista testausta.

Koska asiakkaan näytöt olivat pitkälti lomakemuotoisia, testauksen pääpainoalueeksi valittiin

- lomakkeen peruskäsittely, tietojen näyttäminen, virhekäsittely jne.
- tietojen taulukkolistaus
- välilehtitoiminnallisuus.

PoC-testauksen alustavaksi kestoksi päätettiin varata kolme kalenteriviikkoa per testattava teknologia.

Tutkittaviksi erityisalueiksi tunnistettiin:

- Kuinka käyttöliittymien konversio vanhasta teknologiasta toteutetaan uudelle teknologialle?
- Onko käyttöliittymien konversiota mahdollista automatisoida?
- Kuinka helppoa vanhan arkkitehtuurin liiketoimintaprosessien käyttäminen on uuden teknologian kautta? Kuinka selkeäksi liiketoimintaprosessien rajapinta voidaan toteuttaa?
- Kuinka todellisten näyttöjen ja komponenttien autentikointi ja auktorisointi on toteutettu?
- Kuinka virhe- ja palautetoiminnallisuus on toteutettu?
- Kuinka hyvin teknologia tukee automaattista tietojen konversiota ja validointia?
- Yleisesti, kuinka helppoa näyttöjen toteutus ja testaus on?
- Kuinka helppoa on laatia navigaatiologiikka?
- Kuinka helppoa on rakentaa asiakkaan tarpeet täyttävä näytön komponenttien profilointituki?

Toteutettavat näytöt valittiin asiakkaan käyttöliittymästandardin referenssitoteutuksen esimerkkinäytöistä.

4 Teknologian valinta

Vaatimusmäärittelyssä rajattiin mahdollisia teknologioita lähinnä käytettävän ohjelmointikielen mukaan. Ohjelmointikielen tulisi olla Javan olemassa olevien ja säästettäväksi tarkoitettujen sovellusten osasten mukaisesti. Tällä ei suoranaisesti poissuljettu RIA-tyyppisiä ratkaisuja. Lisäksi vaatimusmäärittelyssä päädyttiin suosimaan avoimen lähdekoodin ratkaisuja.

Teknologian valintaprosessi käynnistyi valinnassa käytettävien kriteerien keräämisellä, jonka jälkeen kriteereille laadittiin painoarvot ja arvosteluasteikot. Tämän jälkeen tutkittiin olemassa olevia, esiehtojen mukaisia teknologioita. Tunnistetuista teknologioista valittiin yhdeksän tarkempaan analyysiin: JSF (2.0), Spring MVC (3.0.7), Wicket (1.5.2), GWT (2.4), Tapestry (5.2.6), Spring Web Flow (2.3.0), Grails (1.3.7), Vaadin (6.7.1) ja Struts (2.2.3.1). Näistä Spring Web Flow paljastui lähinnä navigaatioteknologiaksi Spring MVC:n päälle, joten se pudotettiin vertailusta. Teknologioiden analyysista tuotettiin vertailumatriisi aiemmin laadittujen arvostelukriteerien mukaisesti. Vertailumatriisin tulosten pohjalta oli tarkoitus valita kaksi tai useampaa parhaimmalta vaikuttavaa teknologiaa kolmen viikon koekäyttöön, jonka aikana olisi tuotettu vaatimusmäärittelyn aikana määritelty PoC-sovellus kummallakin teknologialla. Pisteytyksen perusteella valittiin PoC-vaiheeseen kuitenkin vain yksi teknologia, Java EE6 -standardin mukainen JSF2.0. Syynä tähän olivat hyvät pisteet ja standardipohjainen ratkaisu. Lisäksi aikaa olisi koeponnistaa toiseksi parhaalta vaikuttavaa ratkaisua, mikäli PoC-testauksessa havaittaisiin tuotantokäytön estäviä haasteita. PoC-testaus sujui odotettua helpommin, vaikka muutamia haasteitakin huomattiin. PoC-sovelluksen demotilaisuudessa valittiin yksimielisesti käytettäväksi käyttöliittymäteknologiaksi Java EE6 -standardin mukainen JSF2.0.

4.1 Kriteerien valinta

Kriteerien valinta käynnistyi etsimällä netistä käyttöliittymäteknologioiden vertailuja ja niissä käytettyjä kriteerejä. Hieman yllättäen tällaisia vertailuja löytyi verrattain vähän, ja nekin olivat monesti auttamattomasti vanhentuneita. Löytyneille vertailuille löytyi monasti jopa paremmin argumentoituja vastauksia, jotka nollasivat vertailun tulokset. Matt Raiblen [2, s. 17-20] esityksestä löytyi useita varteenotettavia kriteerejä, joita käyttää teknologian valinnassa, vaikka johtopäätöksistä ja perusteluista ei perustaisi-

kaan. Hänen esityksessään esiteltyjä kriteerejä päädyttiinkin käyttämään vertailumatriisin kriteerien pohjana. Matt Raiblen kriteerit on esitelty tulkintoineen ja käyttökelpoisuuksineen liitteessä 2. *Matt Raiblen Web teknologian valintakriteerit*.

Bryce Boe [3] käsitteli artikkelissaan web-teknologiavertailun ongelmaa. Artikkelinsa lopputuloksena hän päätyi käyttämään Google Trendejä ja StackOverflow-palvelua mittaustensa pohjana.

Matt Raiblen karsittua listaa täydennettiin Bryce Boen sekä muutamalla omalla kriteerillä, jonka jälkeen se käsiteltiin ja hyväksytettiin suunnittelupalaverissa. Palaverissa laadittiin myös painoarvot eri kriteereille sekä mietittiin, miten kriteerejä mitattaisiin. Kaikki kriteerit päädyttiin pisteyttämään asteikolla 0-10, jotta saataisiin aikaiseksi hajontaa. Pisteytystä jouduttiin muokkaamaan useiden kriteerien osalta vertailumatriisia täydennettäessä, eikä pisteytys siltikään ole kaikilta osin looginen. Esimerkiksi kirjojen lukumäärässä tietyn raja-arvon yli menevien kirjojen tarjoama hyöty vähenee jyrkästi, jolloin kyseiseen raja-arvoon yltäneiden teknologioiden tulisi saada täydet pisteet. Tämä aiheuttaa vääristymää vertailumatriisissa. Jälkikäteen ajatellen juuri pisteytykseen ja sen mahdollisiin raja-arvoihin olisi pitänyt käyttää enemmän aikaa. Hyväksytty kriteerilista painoarvoineen ja pisteytyksineen löytyy liitteestä 3. *Hyväksytty kriteerilista painoarvoineen ja pisteytyksineen*.

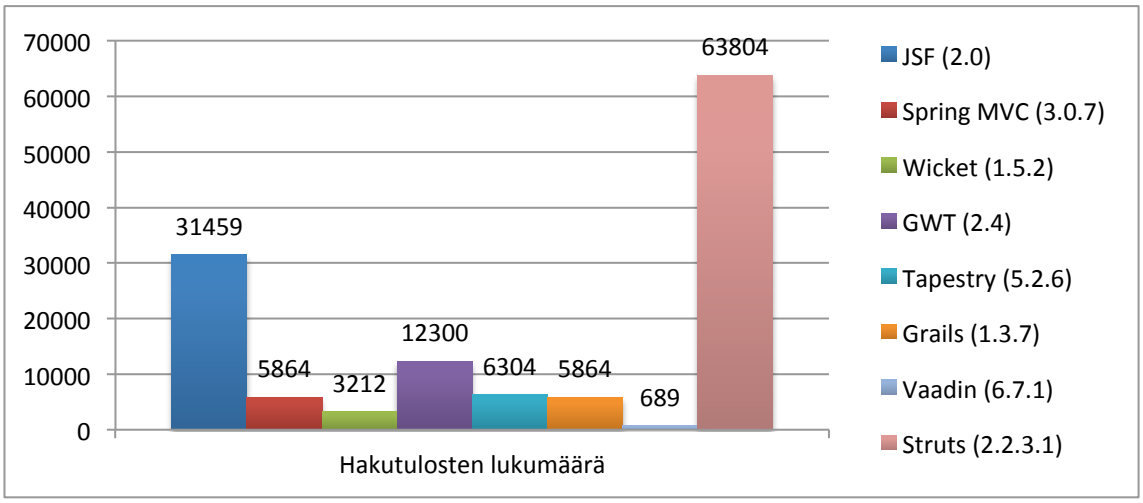
4.2 Vertailumatriisin täyttäminen

Vertailumatriisin täyttämiseen varattiin viikko kalenteriaikaa. Täyttämiseen olisi tullut varata ainakin kaksinkertainen aika - varsinkin, kun kalenteriajasta lähes puolet kului juokseviin asioihin ja palavereihin. Kriteereistä ehdittiin käsitellä vain 12 kahdestakymmenestä. Käsittelemättä jäivät dokumentaation taso, oppimiskynnys, profilointituki, komponenttirajapinnan tuki, testattavuus, skaalautuvuus, tietoturva ja työkalutuki.

Kehittäjien saatavuus

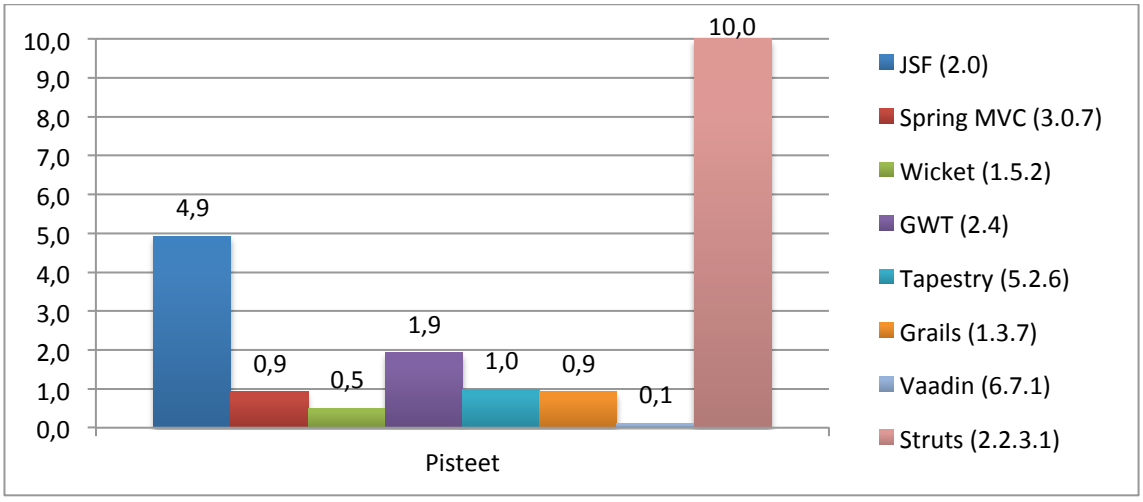
Kehittäjien lukumäärän tutkimiseen käytettiin LinkedIn-palvelua [4], josta etsittiin profiileja, jotka täsmäsivät projektikohtaiseen hakuehtoon. Käytetyt teknologiakohtaiset hakuehdot on esitetty liitteessä 4 *Vertailumatriisin täyttämisessä käytetyt hakuarvot*.

Hakutulokset on esitetty kuvassa 2.



Kuva 2. Kehittäjien saatavuuden hakutulokset

Hakutuloksista valittiin suurin, joka oli Struts. Sille annettiin kymmenen pistettä. Muiden teknologioiden pisteet suhteutettiin eniten hakutuloksia saaneeseen. Hakutuloksien perusteella jaetut pisteet on esitetty kuvassa 3.

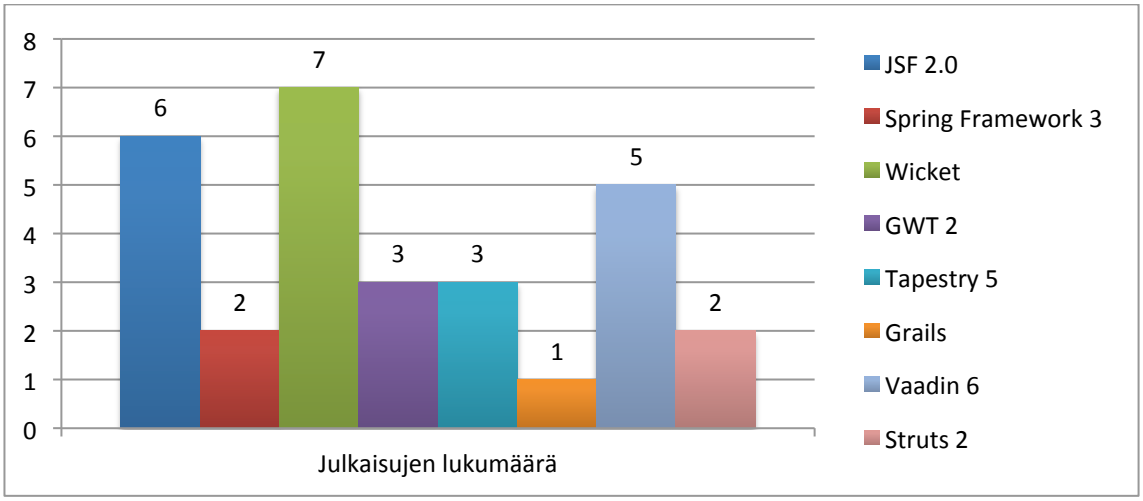


Kuva 3. Kehittäjien saatavuudesta jaetut pisteet

Projektin tila

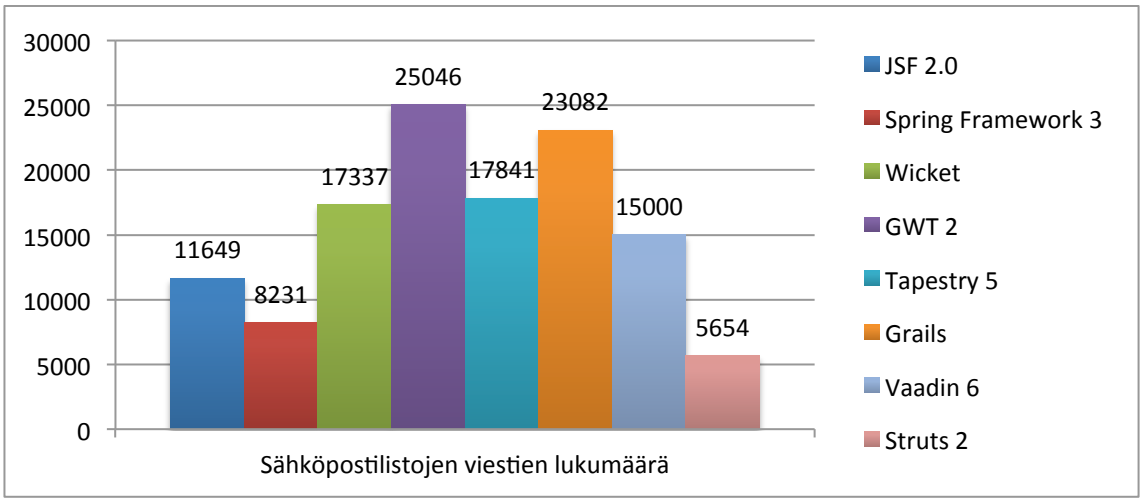
Projektin tilaa tutkittiin laskemalla projektin tuottamat julkaisut vuoden 2011 aikana sekä laskemalla projektin sähköpostilistojen viestien lukumäärä. JSF:n kohdalla ollaan käytetty Mojarra-referenssitoteutuksen julkaisujen lukumäärää. Avoimen lähdekoodin

Apache MyFaces toteutuksen julkaisujen lukumäärä olisi ollut 11. Vaatimen osalta julkaisujen lukumäärä jäi hieman epäselväksi. Projektien tuottamien julkaisujen lukumäärät vuodelta 2011 on esitetty kuvassa 4. [5; 6; 7; 8; 9; 10; 11; 12; 13.]



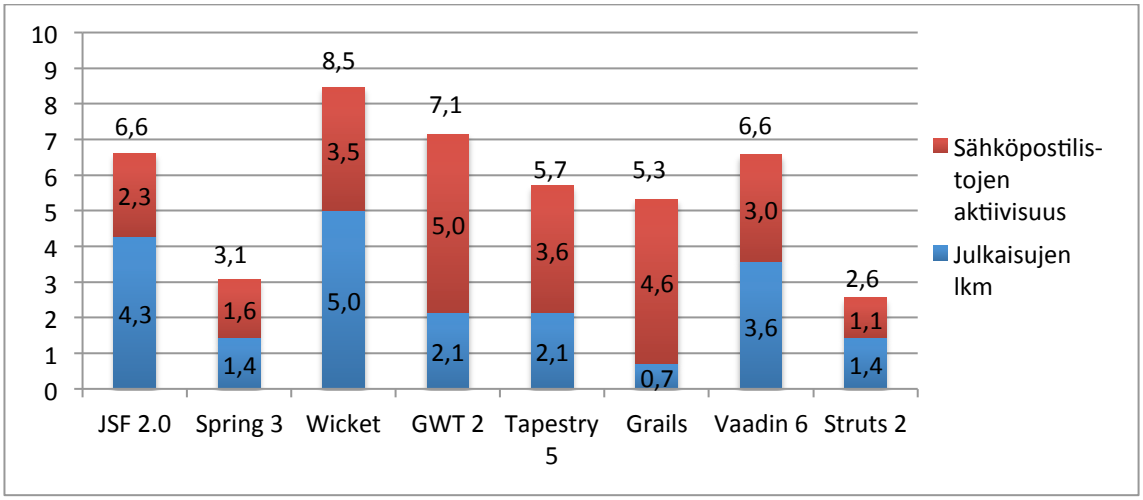
Kuva 4. Projektin tekemien julkaisujen lukumäärä vuonna 2011

Kaikilla projekteilla ei ollut sähköpostilistoja , joten niiden osalta jouduttiin esittämään arviot seuraavin perustein. JSF:n osalta käytettiin Apache MyFaces -projektin sähköpostilistojen aktiivisuutta. Spring-projektit käyttävät vain foorumeita, joten Spring MVC:n osalta vähennettiin ensin käyttöliittymiin liittyvien viestien lukumäärästä Spring Web Flowhun liittyvien viestien määrä. Tämän jälkeen tutkittiin, kuinka moni sivu oli kuluvalta vuodelta, ja esitettiin tämän pohjalta arvio vuoden 2011 Spring MVC liittyvistä viesteistä. Vaadinprojekti käyttää myös pelkästään foorumeita, joten heihin sovellettiin samaa arviointitapaa kuin Spring MVC:n kohdalla. Projektien sähköpostilistojen aktiivisuus vuonna 2011 on esitetty kuvassa 5. [5; 6; 7; 8; 9; 10; 11; 12; 13.]



Kuva 5. Projektin sähköpostilistojen viestien lukumäärä vuonna 2011

Pisteytyksessä jaettiin puolet pisteistä julkaisujen ja toinen puoli sähköpostiviestien lukumäärän mukaan. Julkaisujen osalta Wicket sai viisi pistettä ja sähköpostiviestien osalta GWT. Muiden teknologioiden pisteet suhteutettiin kummassakin ryhmässä eniten pisteitä saaneen lukuarvoon. Projektin aktiivisuuden mukaan annetut pisteet on esitetty kuvassa 6.

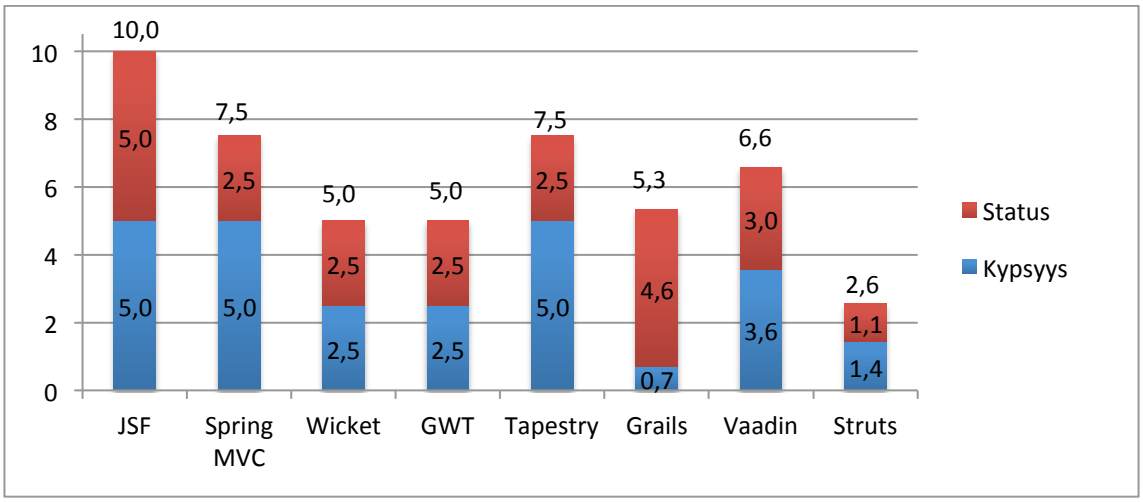


Kuva 6. Projektin tilan mukaan jaetut pisteet

Riski

Riskin osalta teknologioita tutkittiin niiden kypsyyden ja statuksen mukaan [5; 6; 7; 8; 9; 10; 11; 12; 13.]. Puolet pisteistä jaettiin teknologian kypsyyden mukaan. Mikäli pro-

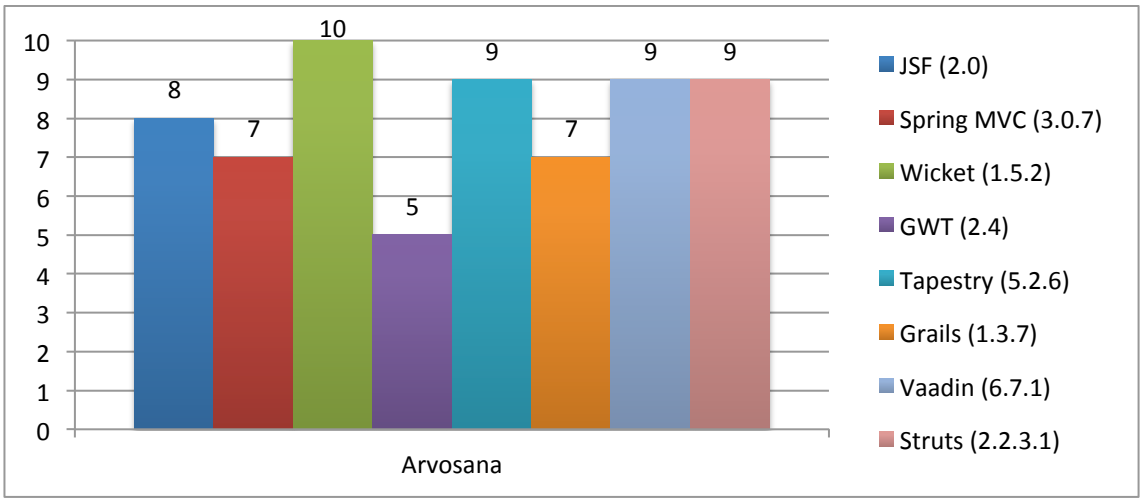
jekti oli edelleen aktiivinen ja se oli kestänyt vähintään kolme vuotta, sai se viisi pistettä. Mikäli projekti oli tuore, mutta jo laajalti käytössä, sai se kaksi ja puoli pistettä. Muuten ei pisteitä annettu. Toinen puolikas pisteistä jaettiin projektin statuksen mukaan. Mikäli kyseessä on standardi, sai se viisi pistettä. Mikäli kyseessä on tunnetun, ison avoimen lähdekoodin yhteisön projekti, esim. Apache, saa se kaksi ja puoli pistettä. Muuten pisteitä ei annettu lainkaan. Projektin riskin mukaan annetut pisteet on esitetty kuvassa 7. [5; 7; 8; 9; 10; 11; 12; 13.]



Kuva 7. Projektin riskin mukaan annetut pisteet

Lokalisaatiotuki

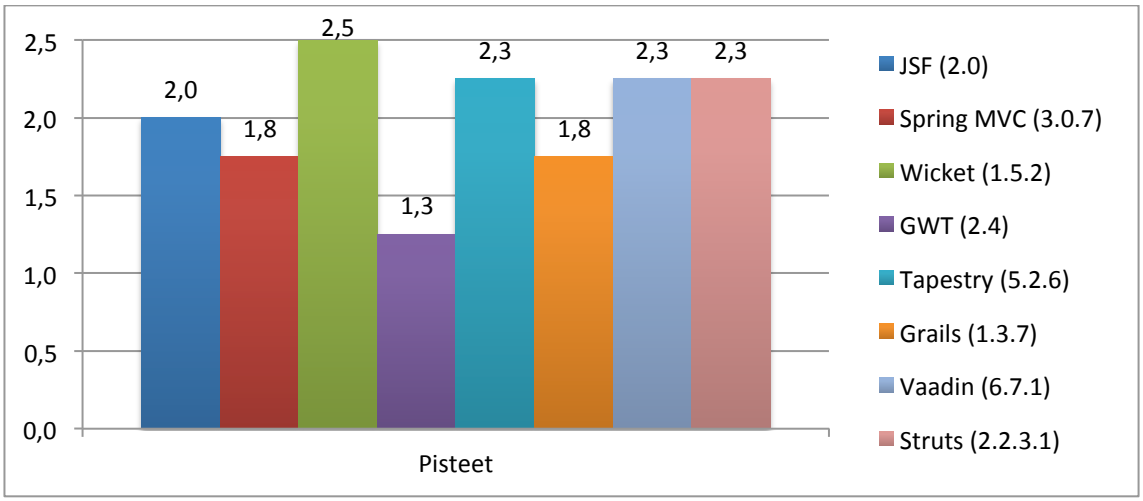
Lokalisaatiotuen pisteytystä varten tutkittiin projektien dokumentaatiota ja perehdytysmateriaaleja lokalisaatiotuen (i18n, l10n) osalta. Mikäli teknologia tuki lokalisaatiota, sai se viidestä kymmeneen pistettä riippuen, kuinka helppoa lokalisaatiotuen käyttäminen oli. Mikäli teknologia ei tukenut lokalisaatiota lainkaan, ei se saanut pisteitä tästä osiosta. Lokalisaatiotuen tasosta annetut arvosanat on esitetty kuvassa 8. [5; 7; 8; 9; 10; 11; 12; 13.]



Kuva 8. Lokalisaatiotuen tasosta annetut arvosanat

Lokalisaatiotuen osalta pisteytys oli hankala, koska kriteeri ei sisältänyt selkeitä kriteereitä jolloin jäi enemmän tilaa subjektiivisille mielipiteille. Arvostelun perustelut on esitetty liitteessä 5 *Vertailumatriisin täyttämisessä käytetyt arvosteluperusteet*.

Lokalisaatiotuen tasosta annetut pisteet on esitetty kuvassa 9. Kuvaajassa esitetyt pisteet on korjattu kriteerille annetulla kertoimella.

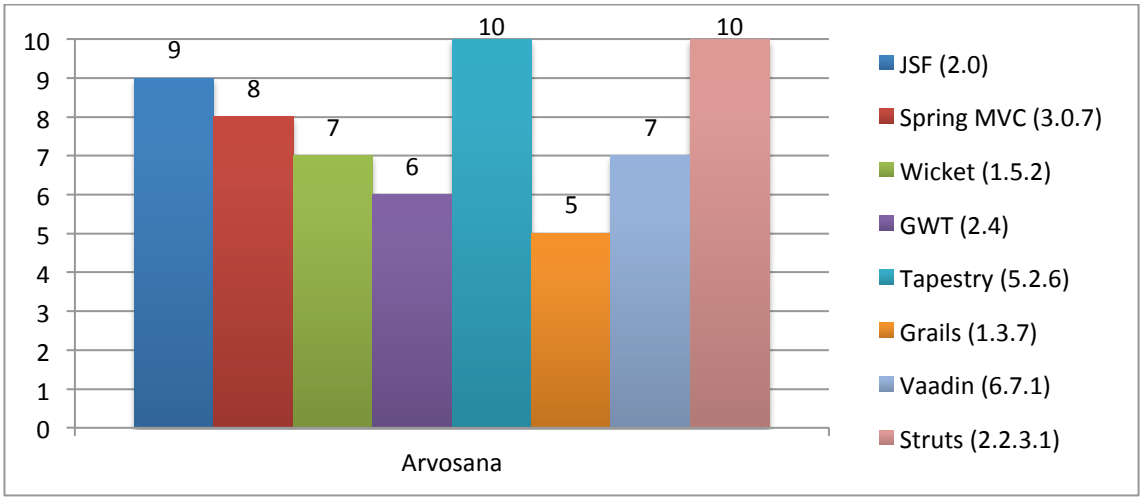


Kuva 9. Lokalisaatiotuen tasosta annetut pisteet

Asiakaspään validaatiotuki

Asiakaspään validaatiotuen pisteytystä varten tutkittiin projektien dokumentaatiota ja perehdytysmateriaaleja asiakaspään validaation ja Ajaxin osalta. Mikäli teknologia tuki

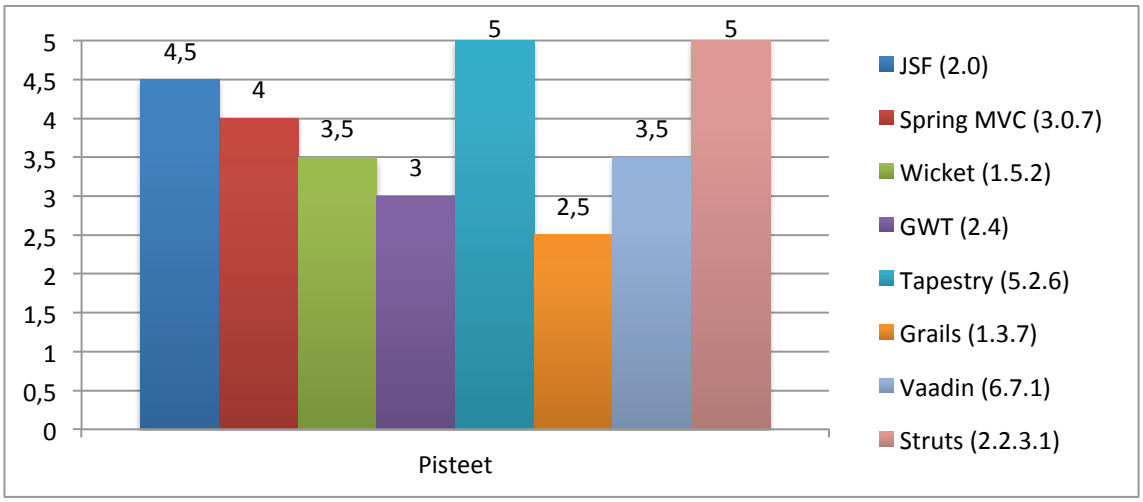
asiakaspään validaatiota, sai se viidestä kymmeneen pistettä riippuen, kuinka helppoa validaation käyttäminen oli. Mikäli teknologia ei tukenut asiakaspään validaatiota lainkaan, ei se saanut pisteitä tästä osiosta. Asiakaspään validaation tasosta annetut arvosanat on esitetty kuvassa 10. [5; 7; 8; 9; 10; 11; 12; 13.]



Kuva 10. Asiakaspään validaation tasosta annetut arvosanat

Asiakaspään validaatiotuen osalta pisteytys oli jälleen hankala, koska kriteeri ei sisältänyt selkeitä kriteerejä, jolloin jäi taas enemmän tilaa subjektiivisille mielipiteille. Arvostelun perustelut on esitetty liitteessä 5 *Vertailumatriisin täyttämisessä käytetyt arvostelu perusteet*.

Asiakaspään validaatiotuen tasosta annetut pisteet on esitetty kuvassa 11. Kuvaajassa esitetyt pisteet on korjattu kriteerille annetulla kertoimella.

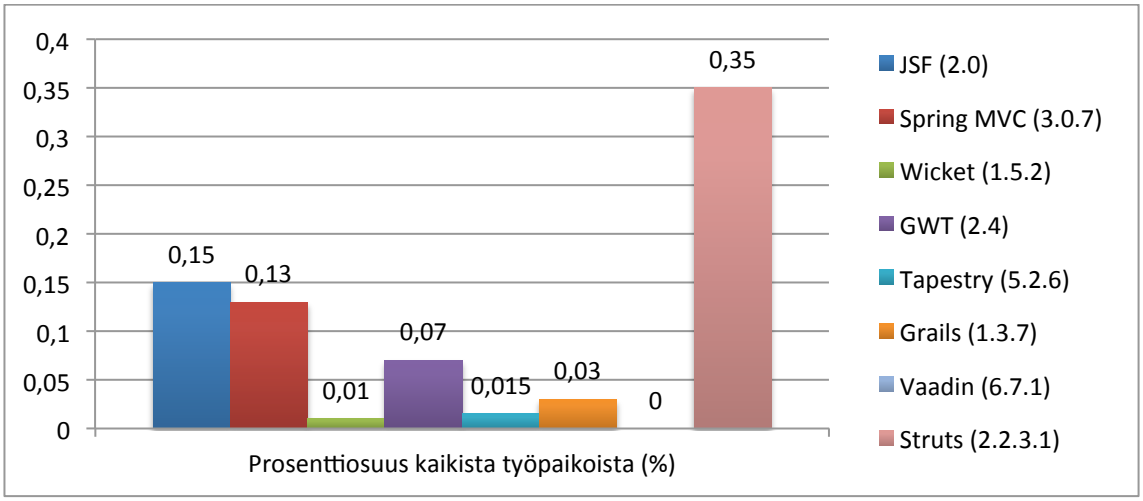


Kuva 11. Asiakaspään validaatiotuen tasosta annetut pisteet

Työpaikkatrendit

Työpaikkatrendien pisteytykseen käytettiin indeed.comia [14], josta poimittiin teknologiakohtaiset hakuehdot täyttävien työpaikkojen prosentuaalinen osuus kaikista työpaikoista. Käytetyt teknologiakohtaiset hakuehdot on esitetty liitteessä 4 *Vertailumatriisin täyttämisessä käytetyt hakuarvot*.

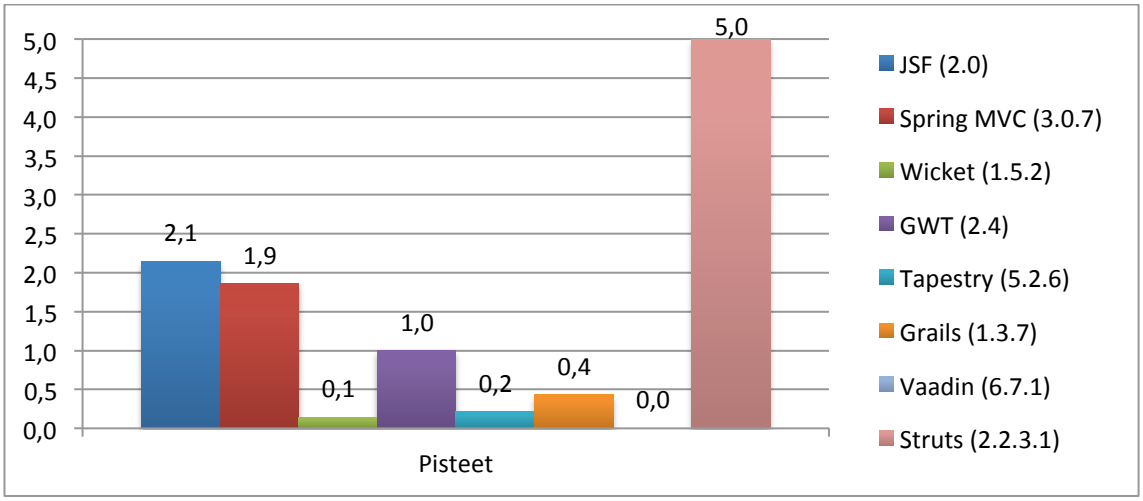
Teknologiakohtaisten työpaikkojen prosenttiosuudet kaikista työpaikoista on esitetty kuvassa 12.



Kuva 12. Teknologiakohtaisten työpaikkojen prosenttiosuudet kaikista työpaikoista

Yksikään teknologioista ei ollut selkeästi menettämässä kiinnostavuuttaan.

Hakutuloksista valittiin suurin, joka oli Struts, ja annettiin sille kymmenen pistettä. Muiden teknologioiden pisteet suhteutettiin eniten hakutuloksia saaneeseen. Hakutuloksien perusteella jaetut pisteet on esitetty kuvassa 13. Kuvaajassa esitetyt pisteet on korjattu kriteerille annetulla kertoimella.

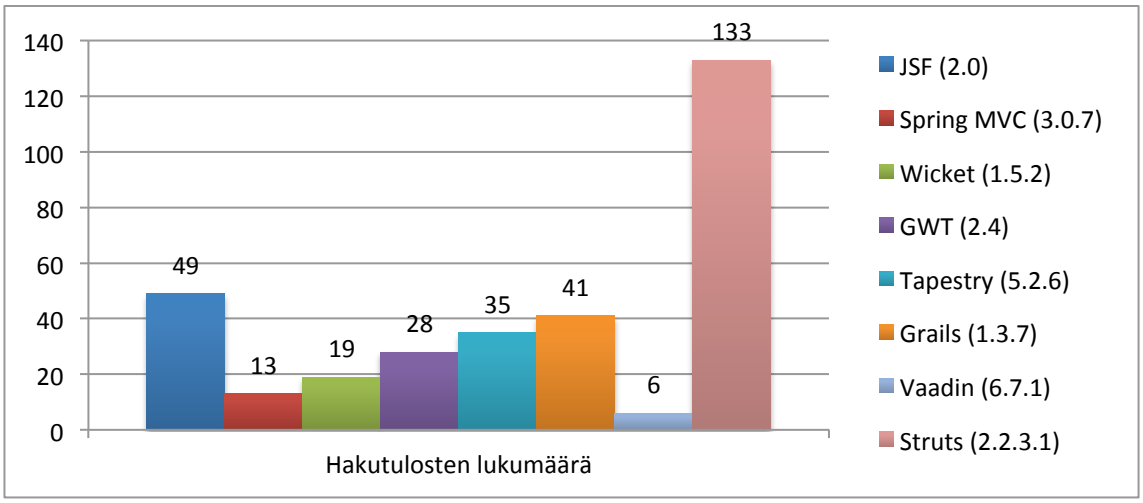


Kuva 13. Työpaikkatrendien perusteella annetut pisteet

Kirjojen lukumäärä

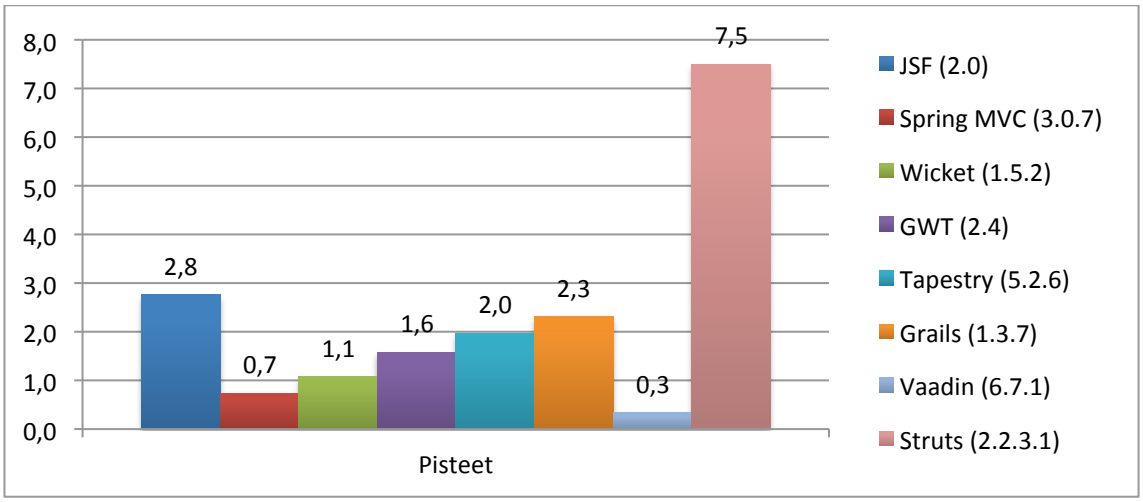
Kirjojen lukumäärien pisteytyksessä käytettiin Amazon.com-verkkokaupan [15] hakupalvelua. Teknologiakohtaisella hakuehdolla suoritettiin haku ja tulokset poimittiin ylös. Käytetyt teknologiakohtaiset hakuehdot on esitetty liitteessä 4 *Vertailumatriisin täyttämisessä käytetyt hakuarvot*.

Amazonin hakumoottori osoittautui melko huonoksi useilla samanaikaisilla hakuehdoilla. Samaten monilla tuloksilla ei teknologian eri versioille tehtyjä kirjoja pystytty rajaamaan kunnolla, mikä aiheutti vääristymää tuloksissa. Varsinkin Struts 1 ja 2 tulokset menivät sekaisin, vaikka teknologiat poikkeavatkin täysin toisistaan, ja ainoa yhdistävä tekijä on nimi [16]. Hakutulosten lukumäärät teknologiakohtaisilla hakuehdoilla on kuvattu kuvassa 14.



Kuva 14. Kirjojen lukumäärä teknologiakohtaisilla hakuehdoilla

Eniten kirjoja oli julkaistu Strutsille, jolle annettiin kymmenen pistettä. Muiden teknologioiden pisteet suhteutettiin tämän mukaan. Hakutuloksien perusteella jaetut pisteet on esitetty kuvassa 15. Kuvaajassa esitetyt pisteet on korjattu kriteerille annetulla kertoimella.

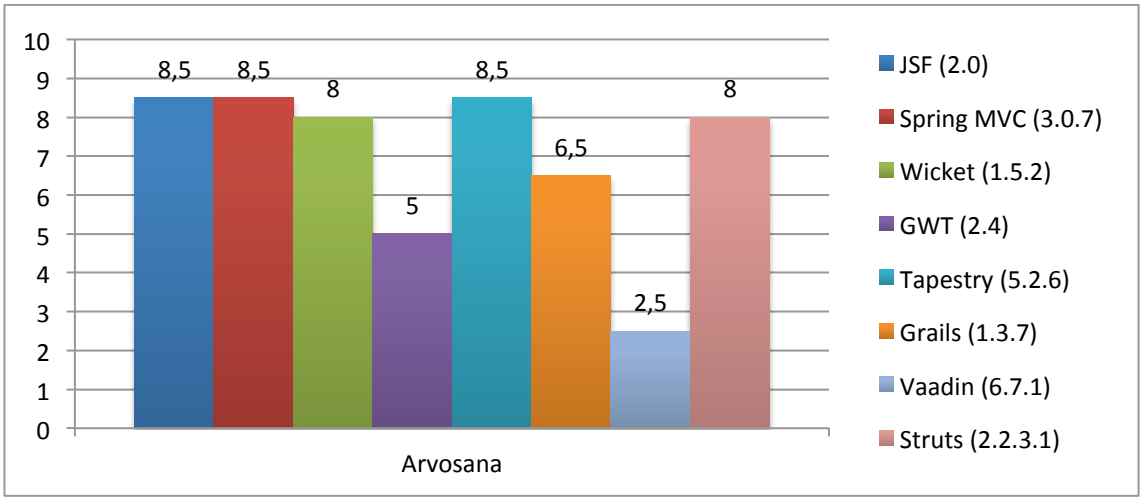


Kuva 15. Kirjojen lukumäärän perusteella annetut pisteet

Suhteellinen pisteytys ei varmaan ollut paras vaihtoehto tämän tyyppiselle kriteerille, koska sata kirjaa lisää ei tee teknologiasta montaa kertaa muita parempaa. Kriteerin kohdalla olisikin pitänyt valita raja-arvot, joiden mukaan pisteet olisi jaettu.

Mallituki

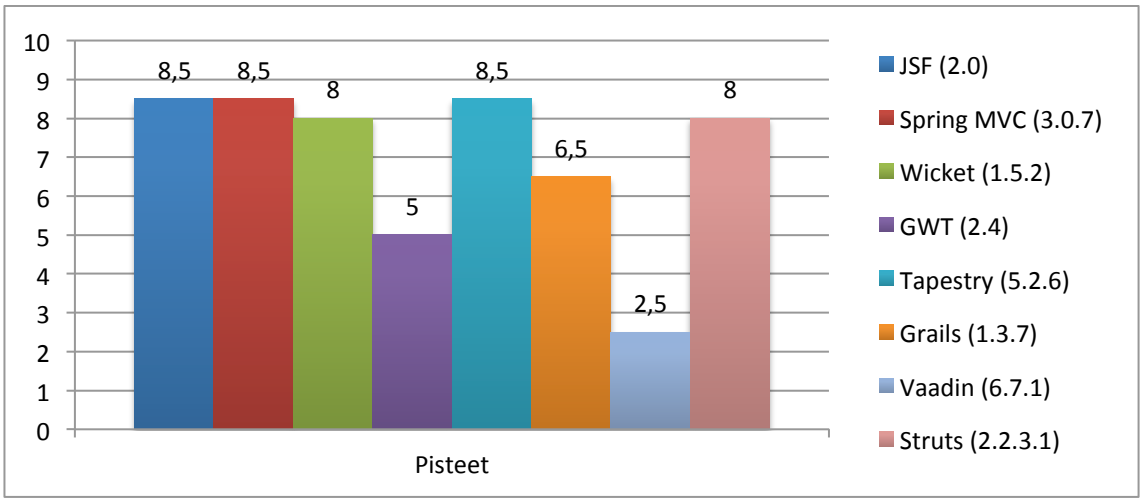
Mallituen pisteytystä varten tutkittiin projektien dokumentaatiota ja perehdytysmateriaaleja mallitiedostojen tuottamisen ja käyttämisen osalta. Mikäli teknologia tuki mallitiedostojen käyttöä, sai se viidestä kymmeneen pistettä riippuen, kuinka helppoa mallien käyttäminen oli. Mikäli teknologia ei tukenut mallitiedostojen käyttöä lainkaan, ei se saanut pisteitä tästä osiosta. Mallitiedostojen tuen tasosta annetut arvosanat on esitetty kuvassa 16. [5; 7; 8; 9; 10; 11; 12; 13.]



Kuva 16. Mallitiedostojen tuen tasosta annetut arvosanat

Mallitiedostojen tuen osalta pisteytys oli jälleen hankala, koska kriteeri ei sisältänyt selkeitä arvosteluperusteita, mikä jätti tilaa subjektiivisille mielipiteille. Arvostelun perustelut on esitetty liitteessä 5 *Vertailumatriisin täyttämisessä käytetyt arvosteluperusteet*.

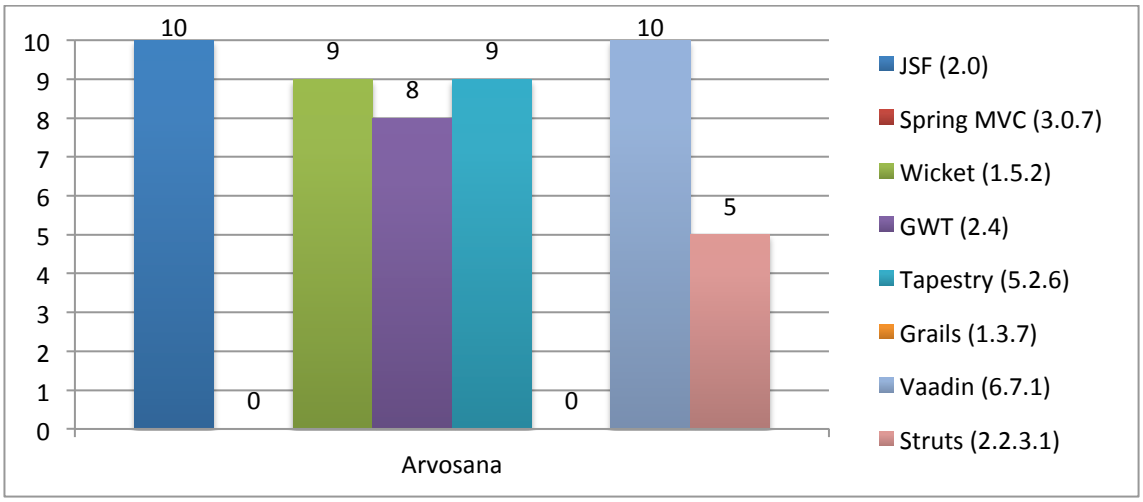
Mallitiedostojen tuen tasosta annetut pisteet on esitetty kuvassa 17. Kuvaajassa esitetyt pisteet on korjattu kriteerille annetulla kertoimella.



Kuva 17. Mallitiedostojen tuen tasosta annetut pisteet

Komponenttituki

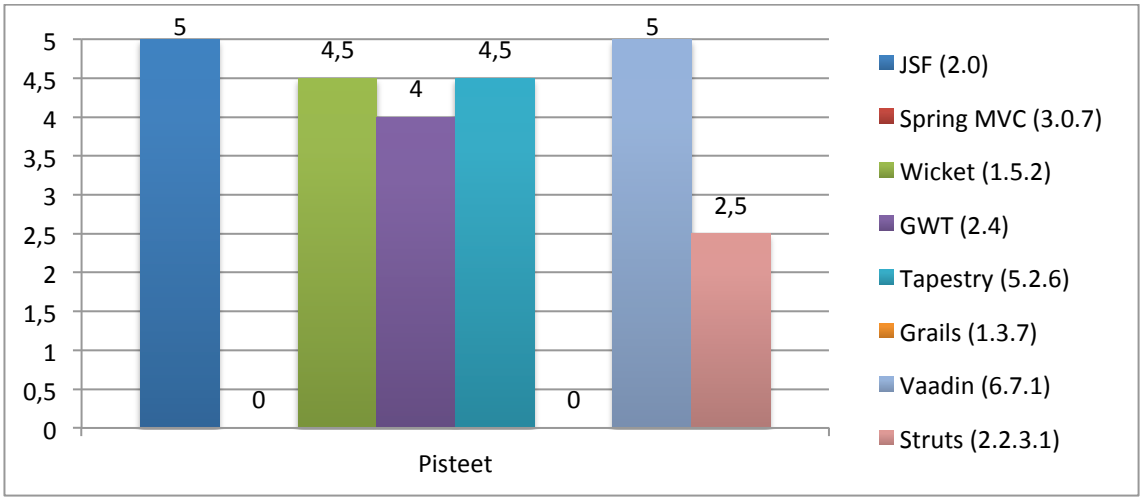
Komponenttituen pisteytystä varten tutkittiin projektien dokumentaatiota ja perehdytysmateriaaleja komponenttien tuottamisen ja käyttämisen osalta. Mikäli teknologia tuki komponentteja, sai se viidestä kymmeneen pistettä riippuen, kuinka helppoa niiden käyttäminen ja tuottaminen oli. Mikäli teknologia ei tukenut komponentteja lainkaan, ei se saanut pisteitä tästä osiosta. Komponenttituen tasosta annetut arvosanat on esitetty kuvassa 18. [5; 7; 8; 9; 10; 11; 12; 13.]



Kuva 18. Komponenttituen tasosta annetut arvosanat

Komponenttituen osalta pisteytys oli jälleen hankala, koska kriteeri ei sisältänyt selkeitä arvosteluperusteita, joka jätti tilaa subjektiivisille mielipiteille. Arvostelun perustelut on esitetty liitteessä 5 *Vertailumatriisin täyttämisessä käytetyt arvosteluperusteet*.

Komponenttituen tasosta annetut pisteet on esitetty kuvassa 19. Kuvaajassa esitetyt pisteet on korjattu kriteerille annetulla kertoimella.

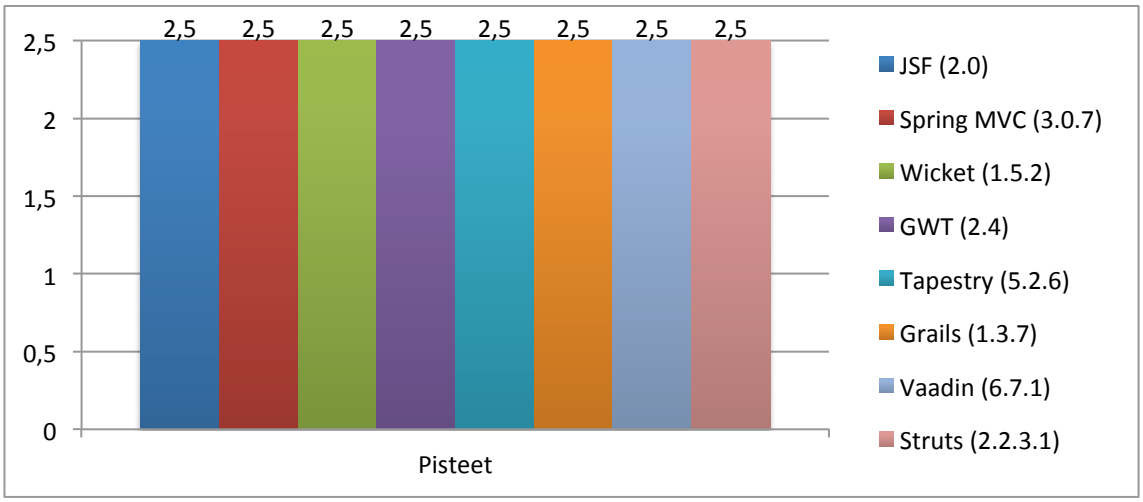


Kuva 19. Komponenttituen tasosta annetut pisteet

Ajax-tuki

Ajax-tuen osalta paljastui, että kaikki valitut teknologiat tukivat sitä natiivisti. Täten kaikki teknologiat saivat arvosanaksi maksimin kymmenen pistettä. [5; 7; 8; 9; 10; 11; 12; 13.]

Ajax-tuen tasosta annetut pisteet on esitetty kuvassa 20. Kuvaajassa esitetyt pisteet on korjattu kriteerille annetulla kertoimella.

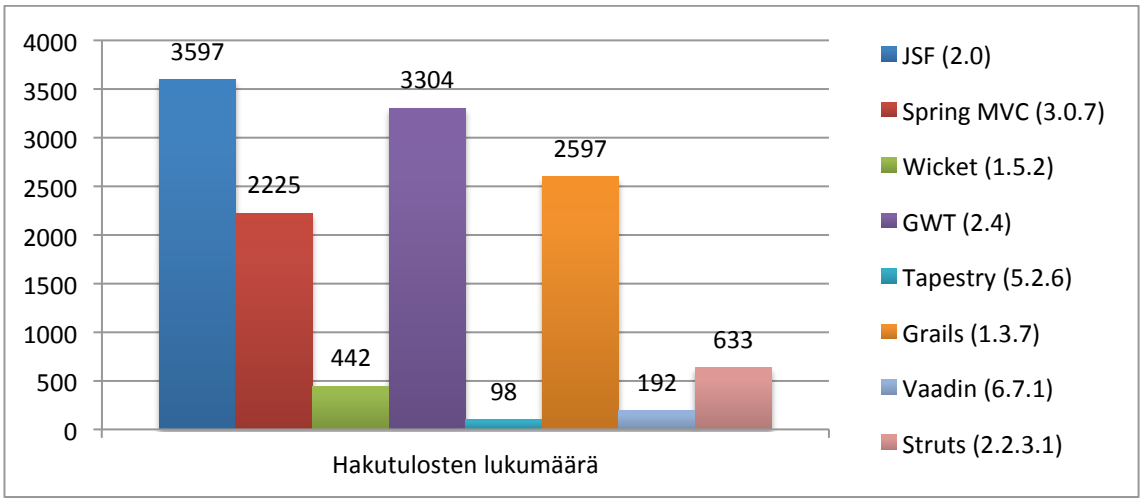


Kuva 20. Ajax-tuen tasosta annetut pisteet

StackOverflow

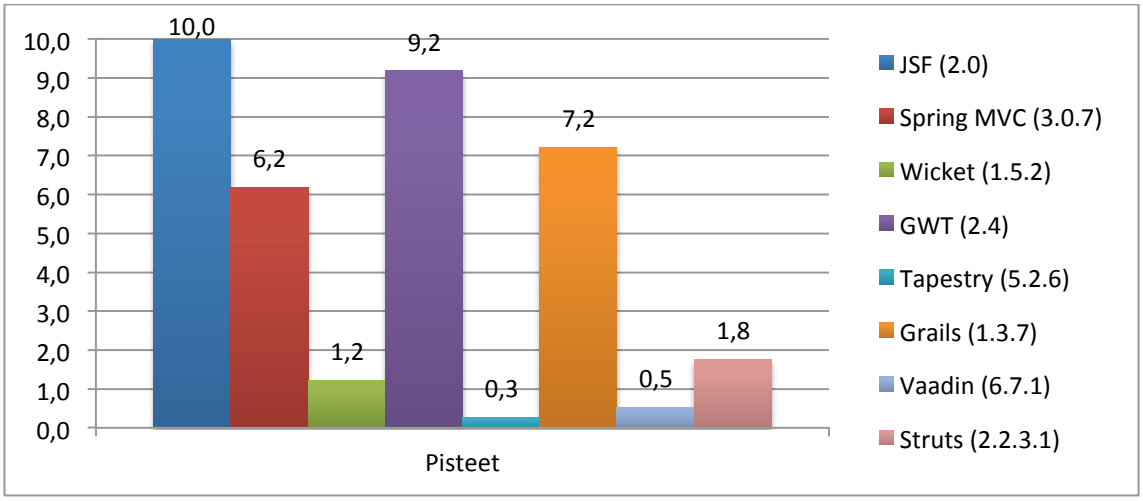
StackOverflow-aktiivisuutta [17] mitattiin hakemalla kysymyksiä teknologiakohtaisilla hakuehdoilla (tag). Käytetyt teknologiakohtaiset hakuehdot on esitetty liitteessä 4 *Vertailumatriisin täyttämisessä käytetyt hakuarvot*.

Hakutulokset on esitetty kuvassa 21.



Kuva 21. StackOverflow-palvelun hakutulokset

Hakutuloksista valittiin suurin, joka oli JSF, ja annettiin sille kymmenen pistettä. Muiden teknologioiden pisteet suhteutettiin eniten hakutuloksia saaneeseen. Hakutuloksien perusteella jaetut pisteet on esitetty kuvassa 22.

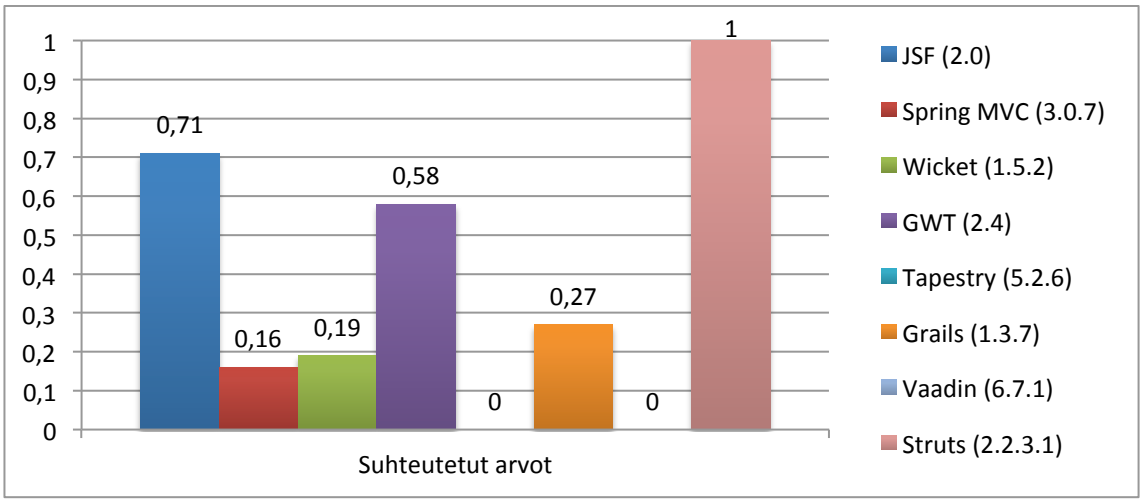


Kuva 22. StackOverflow-aktiivisuudesta jaetut pisteet

Google-trendit

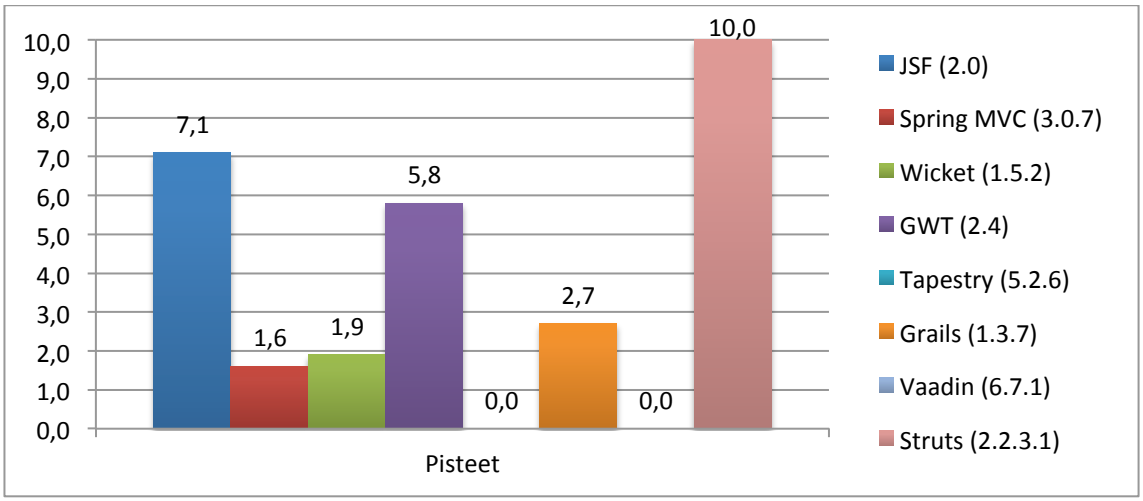
Google-trendejä mitattiin käyttämällä Googlen hakukoneen trendit -osiota [18]. Käytetyt teknologiakohtaiset hakuehdot on esitetty liitteessä 4 *Vertailumatriisin täyttämisessä käytetyt hakuarvot*.

Palvelu antoi vertailussa suosituimmalle hakuehdolle arvon yksi ja muille tähän suhteutetun arvon. Suhteutetut arvot on esitetty kuvassa 23.



Kuva 23. Google Trends -palvelun tulokset

Pisteitä jaettiin palvelun valmiiksi muodostaman kertoimen mukaan kertomalla luku kymmenellä, jolloin suosituin hakutermi sai pisteikseen maksimin kymmenen. Hakutuloksien perusteella jaetut pisteet on esitetty kuvassa 24. Kuvaajassa esitetyt pisteet on korjattu kriteerille annetulla kertoimella.



Kuva 24. Google-trendien perusteella jaetut pisteet

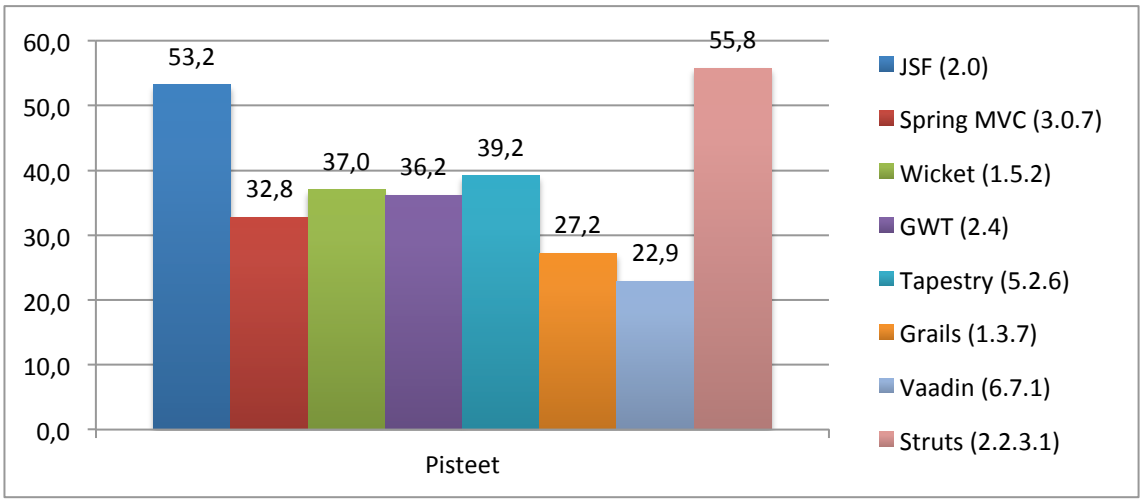
Yhteenveto

Täytetty vertailumatriisi on esitetty taulukossa 4.

Taulukko 4. Täytetty vertailumatriisi

Kriteeri	JSF (2.0)	Spring MVC (3.0.7)	Wicket (1.5.2)	GWT (2.4)	Tapestry (5.2.6)	Grails (1.3.7)	Vaadin (6.7.1)	Struts (2.2.3.1)
Kehittäjien saatavuus	4,9	0,9	0,5	1,9	1,0	0,9	0,1	10,0
Projektin tila	6,6	3,1	8,5	7,1	5,7	5,3	6,6	2,6
Riski	10,0	7,5	5,0	5,0	7,5	2,5	0,0	7,5
Dokumentaation taso	-	-	-	-	-	-	-	-
Oppimiskynnys	-	-	-	-	-	-	-	-
Lokalisaatiotuki	2,0	1,8	2,5	1,3	2,3	1,8	2,3	2,3
Asiakaspään validaatiotuki	4,5	4,0	3,5	3,0	5,0	2,5	3,5	5,0
Työpaikkatrendit	2,1	1,9	0,1	1,0	0,2	0,4	0,0	5,0
Kirjojen lukumäärä	2,8	0,7	1,1	1,6	2,0	2,3	0,3	7,5
Mallituki	8,5	8,5	8,0	5,0	8,5	6,5	2,5	8,0
Komponenttituki	5,0	0,0	4,5	4,0	4,5	0,0	5,0	2,5
Profilointituki	-	-	-	-	-	-	-	-
Komponenttirajapinnan tuki	-	-	-	-	-	-	-	-
Ajax-tuki	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5
Testattavuus	-	-	-	-	-	-	-	-
Skaalautuvuus	-	-	-	-	-	-	-	-
StackOverflow	2,5	1,5	0,3	2,3	0,1	1,8	0,1	0,4
Google trends	1,8	0,4	0,5	1,5	0,0	0,7	0,0	2,5
Tietoturva	-	-	-	-	-	-	-	-
Työkalutuki	-	-	-	-	-	-	-	-
Yhteensä	**53,2**	**32,8**	**37,0**	**36,2**	**39,2**	**27,2**	**22,9**	**55,8**

Kokonaispisteiden tulokset on esitetty kuvassa 25.



Kuva 25. Vertailun kokonaispistemäärät

Teknologiat jakautuivat melko selkeästi kolmeen kastiin. Kärkeä edustivat Struts 2 ja JSF 2.0, hännänhuippuja taasen Vaadin ja Grails. Loput teknologiat olivat keskenään melko tasaväkisiä. Pitää muistaa, ettei tässä vertailtu teknologioiden varsinaista paremmuutta vaan soveltuvuutta asiakkaan tarpeisiin.

4.3 PoC-testaus

Vertailumatriisin tulosten pohjalta oli tarkoitus valita kaksi tai useampi parhaimmalta vaikuttavaa teknologiaa kolmen viikon koekäyttöön, jonka aikana olisi tuotettu vaatimusmäärittelyn aikana määritelty PoC-sovellus kummallakin teknologialla. Vertailun pohjalta kärki oli selkeä: Struts 2 ja JSF 2.0. Koska molemmat ovat paljon käytettyjä teknologioita, löytyi molemmille paljon vastustustakin.

Struts 2:n pisteissä oli monien kriteerien osalta ilmaa, koska tulokset sisälsivät myös aikanaan erittäin suositun Struts 1:n tuloksia, mikä vääristi tuloksia. Tämä ei olisi ollut muuten niin paha asia, mutta Struts 1 ja Struts 2 yhdistävä tekijä on lähinnä nimi. Struts 2 on lähinnä uudelleen nimetty WebWork 2.2 [16].

Keskustelupalstoilla kehotettiin usein välttämään Struts 2:ta ja kerrottiin, että kyseessä on kuoleva teknologia. Yleistä kommenteille oli lähteiden puute, joten nämä kommentit jätettiin huomioimatta. Toisaalta vertailumatriisissa Struts 2 sai huonoimmat pisteet

juuri projektin tilasta, varsinkin virallisten sähköpostilistojen aktiivisuuden osalta, joten jotain perää kommenteissa näyttäisi olevan, mikä huolestutti projektilaisia ja asiakkaan edustajia.

JSF:n osalta kaksi yleisintä vastustajien lähdettä olivat Bruno Borgesin [19] blogipostaus, jossa hän luetteli 10 syytä, miksi vihaa JSF:ää ja Javan ”isän” James Goslingin [20] kommentti yhdessä puheessaan, kuinka hän vihaa JSF:ää.

Blogikirjoituksessaan Borges myöntää, että JSF voi olla tilanteesta tai liiketoimintastrategioista riippuen paras vaihtoehto - varsinkin toimijoille, jotka haluavat löytää mahdollisimman helposti uusia kehittäjiä poistuvien tilalle ja varmistaa sovellustensa ylläpidon. Tällä hän viittaa etenkin isojen toimijoiden, kuten Oracle/Sun, IBM ja Red Hat, satsaamiseen JSF-teknologiaan. Lisäksi blogikirjoituksensa kommenteissa Borges myöntää, että JSF 2.0 on korjannut suurimman osan JSF 1:n ongelmista ja JSF 2.0:n suurin ongelma on JSF 1:n aiheuttama huono maine.

Gosling taasen kertoo puheessaan, kuinka JSF rakennettiin kopioimalla Microsoftin ASF-teknologia. Hän tarkoittaa selkeästi Microsoftin osalta ASP-teknologiaa, mutta JSF ei ole tämän suora vastine, vaan Javan jo vanhentunut JSP-teknologia. Eli Gosling selvästi sekoili lyhenteissä, mutta myytti hänen JSF-vihastaan jäi elämään ja sitä siteerataan melko yleisesti. [21, 4. kommentti.]

Yhteenvetona kumpainenkin kärkiteknologioista oli sinänsä hyväksyttävissä. Struts 2:ta vastaan puhui kuitenkin Struts 1:n vaikutus pisteytyksessä ja projektin vähentyneen aktiivisuuden aiheuttamat huolet. Lisäksi JSF 2.0 oli Java EE -standardi ja standardin versiosta EE6 eteenpäin suositeltu tapa tuottaa web-sovelluksia JSP:n vanhentuessa. Koska tarvittaessa olisi aikaa koeponnistaa toiseksi parhaalta vaikuttavaa ratkaisua, mikäli PoC-testauksessa havaittaisiin tuotantokäytön estäviä haasteita, päätettiin PoC-testaukseen valita vain JSF2.0.

PoC-testaus

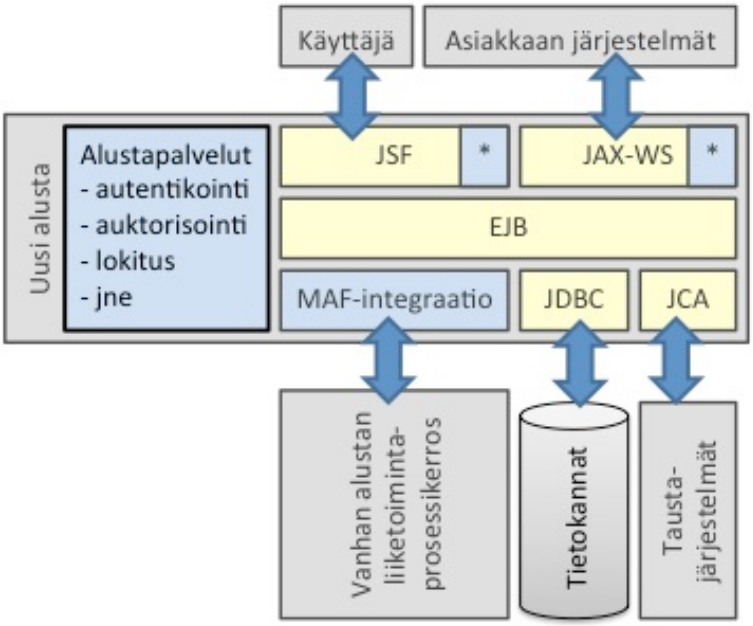
Koska valitun PoC-sovelluksen laajuus oli melko suuri ja kehitysaika kattoi myös hallinnolliset tehtävät, kuten lisenssihakemusten teon ja projektin ulkopuolisten tehtävien hoitamisen, päädyttiin käyttämään kolmen viikon sykliä.

PoC-toteutus alkoi hallinnollisilla tehtävillä, kuten lisenssihakemusten teolla, jonka jälkeen luotiin kehitysympäristöt. Sovelluspalvelimeksi valittiin Apache Tomcat 7.0.22 ja kehitystyökaluksi Eclipse Indigo 3.7.1 Java EE -versio. Ulkoista paketointityökalua ei otettu tässä vaiheessa käyttöön, vaan asennukset toteutettiin eclipsen omalla Tomcat-laajennoksella.

Testauksen aikana saatiin tuotettua vaatimusmäärittelyn yhteydessä määritelty PoC-sovelluksen laajuus. Tuotettua sovellusta ja sen toteutusta esiteltiin liiketoiminta- ja arkkitehtuurivastaaville demotilaisuudessa, jonka yhteydessä valittiin yksimielisesti käytettäväksi käyttöliittymäteknologiaksi Java EE6 -standardin mukainen JSF2.0.

5 Toteutus

Käyttöliittymä- ja SOAP-projektien yhteinen arkkitehtuuri toteutettiin pitkälti Java EE6 -standardin CDI-toiminnallisuuden ympärille. Yhteinen arkkitehtuuri sisältää muun muassa lokitus-, autentikointi- ja auktorisointitoiminnallisuudet sekä projektin tilaan liittyvän resurssien lataamisen ja integraation vanhan sovellusalustan liiketoimintaprosesseihin. Vaikka uudistusprojektien alkaessa ei ollut tarkoituksena tuottaa kokonaan uutta sovellusalustaa vanhan alustan rinnalle, oli yhteisestä arkkitehtuurista käytännössä muodostunut runko sellaiselle. Tämän vuoksi käynnistettiin kolmas projekti viimeistelemään yhteinen arkkitehtuuri valmiiksi sovellusalustaksi ja tuottamaan sille Maven-pohjaisen koostamis- ja riippuvuuksienhallintaratkaisun. Uuden sovellusalustan rakenne on kuvattu kuvassa 26. Java EE -standardin mukaiset osat on kuvattu keltaisella ja asiakkaalle tehdyt räätälöidyt osat sinisellä taustavärillä. Käyttöliittymä- ja SOAP-kanavien sisältämät asiakkaalle toteutetut räätälöidyt sovelluslaajennokset Java EE -standarditeknologioihin on merkitty tähdillä.



Kuva 26. Uuden sovellusalustan rakenne

Käyttöliittymäkerroksen toteutus koostuu kahdesta erillisestä lopputuotteesta: käyttöliittymäarkkitehtuurista ja komponenttikirjastosta. Käyttöliittymäarkkitehtuuri koostuu JSF 2.0:sta ja käyttöliittymäprojektin aikana sen päälle rakennetuista laajennoksista. Yhdessä nämä muodostavat alustan, jonka päälle sovelluskehittäjät toteuttavat käyttöliittymäsovelluksia. Komponenttikirjasto on asiakkaan tarpeisiin rakennettu JSF:n komponenttikirjasto. Komponenttikirjaston komponentit sisältävät valmiiksi asiakkaan käyttöliittymästandardin mukaiset tyylit ja asemoinnit sekä yleisten tietotyyppien tuen. Lisäksi kmponenttikirjasto sisältää tukikomponentteja mm. asemointia ja asetuksia varten.

5.1 Käyttöliittymäarkkitehtuuri

Käyttöliittymäarkkitehtuuri koostuu JSF 2.0:sta ja käyttöliittymäprojektin aikana sen päälle rakennetuista laajennoksista. Laajennoksien pääasiallisena tarkoituksena on sitoa JSF 2.0:n pyynnönkäsittelyn elinkaaren vaiheet yhteisen alustan palveluihin sekä tarjota puuttuvat liiketoiminnalliset vaatimukset.

Käyttöliittymäarkkitehtuuri pohjautuu JSF:n 2.0 versioon, vaikka JSF:n päivitetty versio 2.1 oli saatavilla projektin alkaessa. Versio 2.0:aa voidaan ajaa Java EE5 -ympäristössä ulkopuolisilla kirjastoilla, kun taasen 2.1 vaatii toimiakseen Java EE6 -ympäristön. Asi-

akkaan sovelluspalvelimet oltiin päivittämässä seuraavan vuoden aikana IBM WebSphere 7 -versioihin, jotka olivat vasta Java EE5 -yhteensopivia. JSF 2.0 Java EE5 -mpäristössä toimi muuten hyvin, mutta EL 2.2:n EE6 spesifiset ominaisuudet eivät tietenkään olleet käytettävissä. Näitä olivat muiden kuin getter-metodien ja metodiparametrien käyttö Facelet-tiedostoista taustakomponenttia kutsuttaessa. Lisäksi annotaatiopohjaiset injektiot eivät toimineet taustakomponenteissa. Nämä ongelmat saatiin ratkaistua ottamalla lisäksi käyttöön JBoss:n EL 2.2 -toteutus.

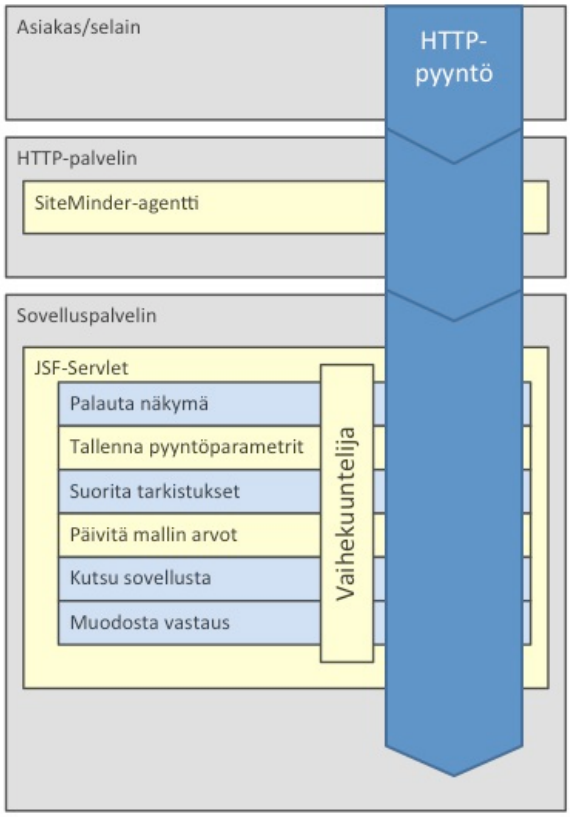
Projektin alussa implementaatioina käytettiin referenssitoteutuksia: Oracle Mojarra JSF:n osalta ja JBoss Weld CDI:n osalta. Kehityksen puolessa välissä huomattiin, että IBM oli ottanut uusimmissa versioissaan WebSphere-sovelluspalvelimesta käyttöönsä Apachen toteutukset: MyFaces JSF:n osalta ja OpenWebBeans CDI:n osalta. Koska asiakas olisi päivittämässä palvelimensa ennemmin tai myöhemmin WebSphere 8 -versioihin, päätettiin ottaa Apachen versiot kirjastoista käyttöön. Nämä ulkopuoliset kirjastot voitaisiin siten myöhemmin pudottaa pois paketoinnista ja varmistuttaisiin, että toiminnallisuudet ja konfiguraatiot toimisivat mahdollisimman hyvin tulevan sovelluspalvelimen omien versioiden kanssa.

Sovelluskehittäjiä varten tuotettiin Java ja Facelet API:t. Näiden API:en tarkoituksena ei ollut piilottaa tai korvata käytettävää teknologiaa, vaan tarjota yhtenäinen rajapinta alustan palveluiden käyttämiseen. Java-puolelle API toteutettiin rajapintana. Sama rajapinta toimii myös Facelet-puolelle, koska rajapinnan toteutus toteutettiin nimettynä (Named) komponenttina, jota voidaan kutsua suoraan Facelet-sivuilta.

Seuraavissa luvuissa on kuvattuna, kuinka käyttöliittymäarkkitehtuurin JSF 2.0 -laajennukset ja palvelut on toteutettu ja kuinka ne toimivat.

5.1.1 Pyynnönkäsittelyn elinkaari

Pyynnönkäsittelyn elinkaari on esitetty korkealla tasolla kuvassa 27. Vaiheet, jotka sisältävät arkkitehtuurin toiminnallisuutta, on kuvattu vaaleansinisellä.



Kuva 27. Pyynnönkäsittelyn elinkaari

SiteMinder-agentti - vastaa http-palvelimella toimialueen laajuisesta autentikaatiosta ja sovellustason auktorisoinnista. Agentti tarkistaa, että käyttäjän toimialueen laajuinen sessio on voimassa ja tarvittaessa ohjaa käyttäjän toimialueen sisäänkirjautumissivulle. Lisäksi tarkistetaan, että käyttäjällä on pääsy pyydettyyn sovellukseen. Tämä toiminnallisuus on osa asiakkaan http-infrastruktuuria.

JSF-servlet - vastaa JSF:n mukaisesta pyynnönkäsittelyn elinkaaren hallinnasta. Tämä toiminnallisuus on osa JSF-standardia.

Vaihekuuntelija - JSF:n elinkaarenvaihe-kuuntelija, joka kuuntelee JSF-servletin lähettämiä tapahtumia pyynnön siirtyessä elinkaaren vaiheesta toiseen. Tämä kuuntelija delegoi kaikki tapahtumat arkkitehtuurin elinkaarenhallinnasta vastaavalle komponentille. Yhden kuuntelijan malliin päädyttiin, koska JSF ei mahdollista kunnollista kuuntelijoiden järjestelemistä ja esimerkiksi autentikoinnista vastaavan kuuntelijan halutaan saavan suoritusvuoronsa ennen muita kuuntelijoita.

Palauta näkymä - Luo tai palauttaa JSF:n palvelinpään komponenttipuun (View) muistiin, joka vastaa asiakaspäässä näytettävää sisältöä. Tämä toiminnallisuus on osa JSF-standardia.

Toteutus laajentaa tätä vaihetta heti näkymän palautuksen jälkeen. Ensin tarkistetaan käyttäjän järjestelmän sisäinen sessio tai alustetaan uusi sessio - mikäli sessiota ei vielä ole aloitettu. Mikäli sessio ei ole voimassa tai saadut http-otsikkotiedot eivät vastaa session tietoja, invalidoidaan sessio ja aloitetaan uusi. Kun sessio on alustettu, pyyntö auktorisoidaan ja tarvittaessa ohjataan käyttäjä virhesivulle. Mikäli pyyntö sisältää salatun URL-parametrin, puretaan parametrin salaus. Lopuksi lisätään käyttäjän, käyttäjän kohdeorganisaation ja käsiteltävän kohteen jne. pyyntökohtaiset tiedot pyyntökontekstiin.

Tallenna pyyntöparametrit - Tallentaa asiakkaan lähettämät tiedot palvelinpään komponenttipuun komponenteille talteen jatkokäsittelyä varten. Tämä toiminnallisuus on osa JSF-arkkitehtuuria.

Suorita tarkistukset - Suorittaa tarkistukset ja tietotyyppien konversiot saapuneelle datalle. Tämä on osa JSF-standardia.

Toteutus laajentaa tätä vaihetta tarjoamalla omia tarkistimia ja konverttereita käyttöön. Nämä tarkistimet ja konvertterit on integroitu suoraan kmponenttikirjaston komponenttien käyttöön tyyppi-attribuutin kautta, mutta koska kyseessä on JSF-standardin mukaisia komponentteja, voi niitä käyttää myös muiden käyttöliittymäkomponenttien kanssa. Komponenttikirjaston komponentit lisäävät tarvittaessa tarkistuksissa tai konversiossa tapahtuneista poikkeuksista virheviestit automaattisesti komponenttipuuhun.

Mikäli vaiheen suorituksen aikana havaitaan poikkeuksia, keskeytetään pyynnön käsittely ja siirrytään vastauksen muodostamiseen. Vastauksen muodostuksessa käyttäjälle palautetaan sivu, jolla hän oli, virhetietojen kera.

Päivitä mallin arvot - Päivittää asiakaspään lähettämät tarkistetut ja konvertoidut arvot palvelinpään komponenttipuun komponenttien uusiksi arvoiksi. Tämä toiminnallisuus on osa JSF- arkkitehtuuria.

Kutsu sovellusta - Kutsuu pyynnön käsittelemiseksi tarvittavaa sovelluslogiikkaa ja tarvittaessa navigoi uudelle sivulle. Tämä toiminnallisuus on osa JSF-arkkitehtuuria.

Tässä vaiheessa JSF-arkkitehtuuri kutsuu sovelluslogiikkaa, mikäli pyyntö sisälsi toiminnon. Toteutus tarjoaa tämän vaiheen aikana sovelluskehittäjille käyttöön vain työkaluja ja apusovelluksia, joita he voivat käyttää tarvittaessa.

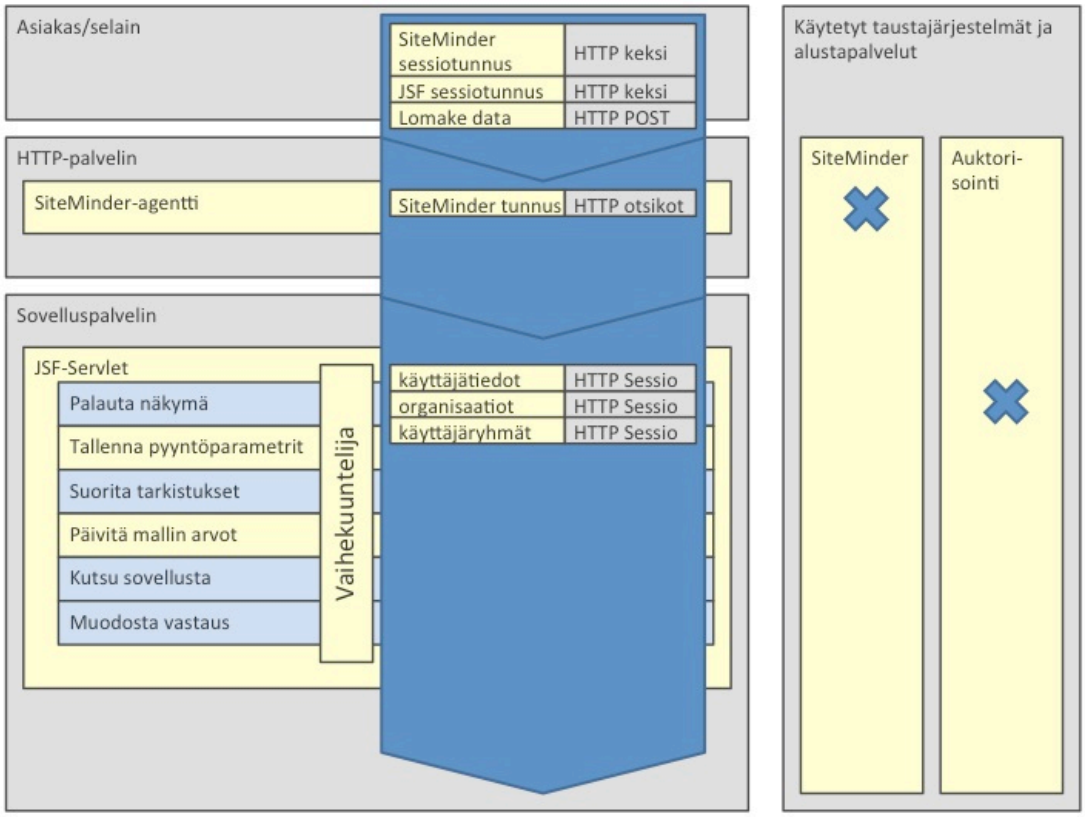
Muodosta vastaus - Muodostaa vastauksen kutsumalla komponenttipuun komponenttien muuntajia sekä tallentaa komponenttipuun tilan. Tämä toiminnallisuus on osa JSF-arkkitehtuuria.

Toteutus laajentaa tätä vaihetta tarjoamalla kehittäjien käyttöön Komponenttikirjaston ja erinäisiä sivupohjia. Komponenttikirjaston komponentit, JSF:n standardikomponentit ja sivupohjien mallipohjat tuottavat asiakkaalle lähetettävän XHTML-datan. Komponenttikirjaston komponentit käyttävät käyttäjälle näytettävien tietojen profiloinnissa tetoturvapalvelua.

5.1.2 Käyttäjän sessiot, autentikointi ja auktorisointi

Sovelluksen autentikointi ja auktorisointi toteutettiin osana JSF:n pyynnönkäsittelynelinkaarta, koska tällöin kaikki tarvittavat kontekstit ovat käytössä. Palvelunestohyökkäyksiä ei tarvinnut ottaa tässä kohtaa huomioon, koska sovellukset toimivat SiteMinderin suojaamassa aliverkossa. Lisäksi sovellusympäristö toimii omassa suljetussa verkkoympäristössään, jonne käyttäjät ottavat yhteyttä VPN:n avulla.

Arkkitehtuurin pyynnönkäsittelyn aikana autentikointiin ja auktorisointiin käytettävät taustajärjestelmät ja -palvelut sekä näihin liittyvät tiedot eri vaiheiden aikana on esitetty kuvassa 28. Pyynnönkäsittelyn vaiheet, joissa käytetään taustajärjestelmiä tai alustan palveluita, on merkitty ruksilla kyseessä olevan järjestelmän tai palvelun kohdalle.



Kuva 28. Pyynnönkäsittelyn aikana käytettävät taustajärjestelmät ja -palvelut

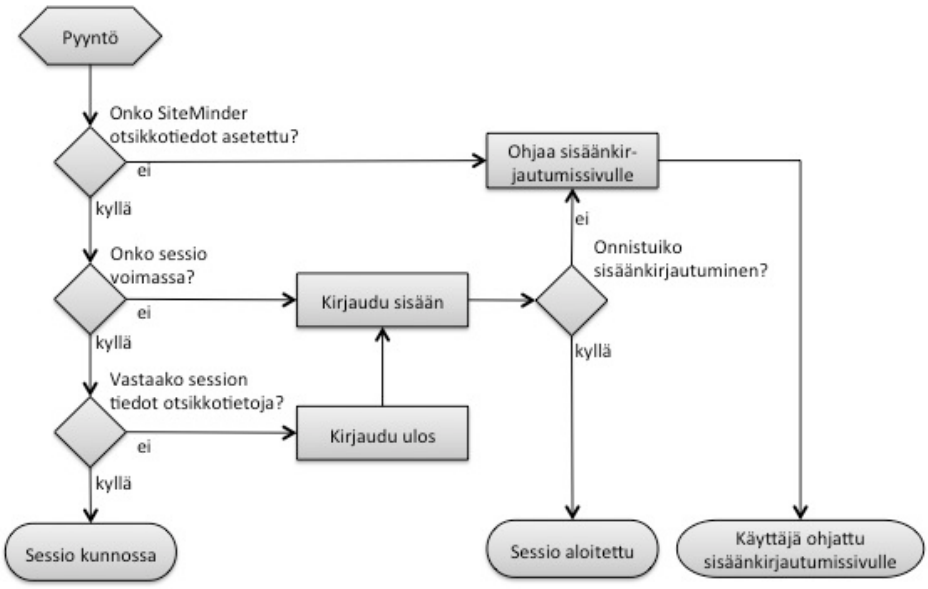
Kuvasta havaitaan, että käyttäjällä on itse asiassa kaksi erillistä sessiota. Ensimmäinen on toimialueen laajuinen SiteMinder-sessio, joita käyttäjällä on aktiivisena vain yksi kerrallaan. Toisena ovat sovelluskohtaiset http-sessiot, joita hallitsee JSF-arkkitehtuuri. Näitä sessioita käyttäjällä on yksi jokaista sovellusta kohden. Molemmat sessiotyypit tallentavat sessiokohtaiset tikettinsä selaimen kekseihin.

Http-palvelimella SiteMinder-agentti sieppaa saapuneen http-pyynnön ja tarkistaa, onko käyttäjän SiteMinder-sessio voimassa ja validi. Mikäli käyttäjän sessio ei ole kunnossa, agentti ohjaa käyttäjän toimialueen sisäänkirjautumissivulle ja pyynnön käsittely keskeytyy. Mikäli sessio on kunnossa, agentti pyytää SiteMinder-palvelimelta käyttäjän tiedot. Tämän jälkeen agentti tarkistaa käyttäjän oikeudet käyttää pyydettyä sovellusta ja tarvittaessa evää pääsyn sovellukseen. SiteMinder on asetettu auktorisoimaan pääsy vain sovellustasolla, esim. http://www.domain.fi/**sovellus1**/. Sovellukset vastaavat itse sisäisestä auktorisoinnistaan. Tällä tavoin sovellusmuutosten tekeminen on huomattavasti helpompaa ja sovelluksesta voi olla useita versioita yhden toimialueen eri testipalvelimilla. Lopuksi agentti tallentaa käyttäjän tiedoista SiteMinder-tunnisteen, session-

tunnuksen ja käyttäjätunnuksen http-pyynnön otsikkotietoihin sovellusten käytettäväksi.

Sovelluspalvelimella arkkitehtuurin elinkaarenhallinnasta vastaava komponentti saa http-pyynnön JSF:n *palauta näkymä* -aiheessa. Pyynnölle suoritetaan tässä vaiheessa sovellustason sisäänkirjautuminen ja auktorisointi.

Sisäänkirjautumisen aikana toteutus tarkistaa, onko käyttäjällä aktiivista sessiota. Mikäli käyttäjän sessio on voimassa ja sen tiedot vastaavat SiteMinderin asettamia otsikkotietoja, siirrytään auktorisointivaiheeseen. Mikäli käyttäjän sessio ei ole voimassa tai sen tiedot eivät vastaa SiteMinderin asettamia otsikkotietoja, luodaan käyttäjälle uusi sessio. Sessiota varten haetaan SiteMinder-palvelimelta käyttäjän tiedot, käyttäjäryhmät ja organisaatiot. Sisäänkirjautuminen on esitetty tarkemmin aktiviteettikaaviona kuvassa 29.



Kuva 29. Sisäänkirjautumisen aktiviteettikaavio

Auktorisointivaiheessa tarkistetaan käyttäjän pääsyoikeudet pyydettyyn resurssiin. Mikäli käyttäjällä ei ole oikeuksia, ohjataan hänet virhesivulle ja pyynnön käsittely katkeaa. Auktorisointi tapahtuu käyttäen alustan aktorisointipalvelua, joka käyttää XML-muotoista auktorisointidataa. Auktorisointidata pohjautuu vanhan arkkitehtuurin auktorisointipalvelun käyttämiin tiedostoihin, koska mitään vahvaa auktorisointistandardinotaatiota ei tunnistettu eikä niitä katsottu tarpeelliseksi muokata väkisin.

5.1.3 Kehitysympäristöjen autentikointi

Sovelluskehittäjien omissa työasemissa ei ollut asennettuna koko http-infrastruktuuria, eikä siten myöskään SiteMinder-agentteja. Tämän vuoksi toteutukselta puuttui SiteMinderin asettamat otsikkotiedot sekä toimialueen laajuinen sisäänkirjautumissivu. Täten tuotannon autentikointiratkaisua ei voitu käyttää sellaisenaan kehittäjien omilla työasemilla.

Tämä olisi voitu ratkaista kahdella tavalla. Ensimmäinen oli tehdä erillinen autentikointiratkaisu kehitysympäristöihin, kuten vanhassa arkkitehtuurissa oli tehty. Tämä johtaisi kuitenkin ympäristökohtaiseen autentikointiratkaisuun ja erottaisi tuotannon toiminnallisuutta turhaan testivaiheen toiminnallisuudesta. Toinen vaihtoehto oli lisätä SiteMinder-agenttia jäljittelevä toiminnallisuus, jolla voitiin syöttää tarvittavat otsikkotiedot http-pyynnöille ja tarjota kehittäjille sisäänkirjautumissivu. Päädyimme jälkimmäiseen vaihtoehtoon.

Sisäänkirjautumissivu toteutettiin JSF:llä toimialueen sisäänkirjautumissivua vastaavaksi. Sisäänkirjautumissivun taustakomponentti käyttää autentikointiin SiteMinder-palvelinta SOAP-kutsulla, tallentaa SiteMinderin sessiotunnisteen käyttäjän selaimen kekseihin ja muutenkin pyrkii matkimaan SiteMinder-agentin ulkoista toiminnallisuutta.

Varsinainen kehitysympäristöjen autentikointiratkaisu rakennettiin käyttäen CDI:n toiminnallisuuksia, kuten stereotyypit ja vaihtoehtoiset toteutukset. Autentikointitoiminnallisuus erkautettiin omaan komponenttiinsa, jolle luotiin kehityksenaikaisen SiteMinder-agenttia jäljittelevän toiminnallisuuden sisältävä aliluokka. Pääluokka toimii tuotannon autentikointiratkaisuna, mutta kehitysympäristössä sen tilalle ladataan aliluokka käyttöön käyttäen CDI:n toiminnallisuuksia. Tällöin kehitysaikainen autentikointitoiminnallisuus on mukana sovelluspaketissa, mutta se on poissa päältä ja käytössä vain, mikäli niin erikseen halutaan. Tämä vähentää paketoinnin monimutkaisuutta ja toimivaksi testattu sovelluspaketti voidaan siirtää seuraavaan testausvaiheeseen sellaisenaan.

5.1.4 Session hallinta ja näkyvyysalueet

Alustan sessiotoiminnallisuus pohjautuu Java EE6 -standardin CDI-toiminnallisuuteen eikä sen vuoksi JSF:n omia näkyvyysalueita otettu käyttöön. JSF:n näyttökohtainen näkyvyysalue nähtiin kuitenkin sovelluskehittäjille helppokäyttöiseksi ja sen vuoksi päädyttiin rakentamaan sitä vastaava oma näkyvyysalue CDI-laajennoksena. Suositellut näkyvyysalueet ovat täten pyyntö, näyttö, keskustelu, sessio, sovellus tässä esitetyssä järjestyksessä.

Pyyntökohtaista näkyvyysaluetta suositellaan käytettäväksi aina kun mahdollista. Tämä auttaa pitämään http-session koon mahdollisimman pienenä, mikä taasen vähentää sovelluspalvelimen muistinkäyttöä. Pyyntökohtaiselle näkyvyysalueelle laitetut komponentit elävät yhden http-pyynnön käsittelyn ajan. Tämä mahdollistaa tietojen välittämisen näytöltä taustakomponentille ja takaisin. Näkyvyysalueen lyhyt elinkaari tuo toisaalta omia ongelmiaan näytöillä, jotka haluaisivat käyttää taustakomponenttia välimuistina, esim. näytöillä, jotka tarvitsevat tietoja useista taustajärjestelmistä ja tietokannoista. Tällöin on suositeltavaa käyttää näyttökohtaista näkyvyysaluetta.

Näyttökohtaisella näkyvyysalueella olevat komponentit elävät niin kauan, kuin käyttäjä pysyy samalla näytöllä. Heti kun käyttäjä navigoi uudelle näytölle, komponentti siivotaan. On huomattava, että mikäli käyttäjällä on useita välilehtiä auki samalle sivulle, on jokaisella näillä oma sessionsa ja elinkaarensa. Näkyvyysalue on omiaan, mikäli halutaan käyttää taustakomponentteja välimuistina, esim. kun näytetään tietoja useista tietokannoista ja taustajärjestelmistä.

Keskustelukohtaista näkyvyysaluetta voidaan käyttää ohjattujen sivusiirtymien toteuttamiseen. Näkyvyysalue eroaa näyttökohtaisesta näkyvyysalueesta lähinnä manuaalisen elinkaarenhallinnan osalta. Keskustelu täytyy aloittaa ja päättää manuaalisesti, ja se voi elää useiden näyttösiirtymien ajan. Käyttäjällä voi olla useita samanaikaisia keskusteluja voimassa - jopa samalle sivulle eri välilehdillä.

Sessiotasoinen näkyvyysalue toimii kuin klassinen http-sessio. Komponentit siivotaan sessiotasoiselta näkyvyysalueelta käyttäjän http-session mukana.

Sovellustason näkyvyysalue on jaettu kaikkien käyttäjien kesken ja instansseja on vain yksi per sovellus per sovelluspalvelin.

5.1.5 Arkkitehtuurin käyttämät näkyvyysalueet

Arkkitehtuuri käyttää näkyvyysalueita seuraavilla tavoilla.

Sovellustaso

- tietoturvapalvelu: XML-tiedostoista luettujen käyttäjäoikeuksien välimuisti

Sessiotaso

- tietoturvapalvelu: kirjautuneen käyttäjän tiedot
- tietoturvapalvelu: kirjautuneen käyttäjän käyttäjäryhmät
- tietoturvapalvelu: kirjautuneen käyttäjän organisaatiot
- käsiteltyjen sovelluskohtaisten kohteiden historia

Pyyntötaso

- aktiivinen kohdeorganisaatio
- aktiivinen sovelluskohtainen kohde
- käyttäjän lokalisaatioasetus

5.1.6 Tietojen välitys näyttöjen ja sovellusten välillä

Tietoja voidaan välittää useilla tavoilla näyttöjen ja sovellusten välillä. Alun perin tietojen välityksessä päätettiin käyttää JSF:n näyttöparametreja ja pakotettua uudelleenohjausta. Tällöin saataisiin toteutettua tietojen välitys Post/Redirect/Get-suunnittelumallin (PRG) mukaisesti [22]. PRG-malli voidaan mieltää puristisena mallina, jossa http:n POST-operaatiot rajoitetaan tietojen lähetykseen ja GET-operaatiot näytettävän datan hakuun. Mallin mukaan lomakkeen tiedot lähetetään ensin POST-operaatiolla, jonka jälkeen palvelin lähettää selaimelle uudelleenohjauskomennon. Selain tekee sitten GET-operaation uudelleenohjauksessa määriteltyä URL:a käyttäen, mikä sisältää tietojen näyttämiseen tarvittavat parametrit.

PRG-malli tuottaa perinteiseen tapaan nähden, jossa POST-operaatio palauttaa suoraan näytettävän datan, enemmän palvelinkutsuja ja lisää siten näyttöjen vasteaikoja ja palvelimien kuormitusta. Toisaalta malli ratkaisee useita ongelmia. Näitä ovat esimerkiksi kaksinkertainen lähetys, selaushistorian käyttö eteen-/taaksepäin, kirjanmerkkien tuottaminen tietyn datan tilanteen näyttämiseen tietyllä sivulla ja välimuistipalvelimen käytön tuki.

Liiketoimintasovellusten edustajat esittivät kuitenkin huolensa tästä mallista, koska tällöin käyttäjä voisi esimerkiksi henkilötietoja käsiteltäessä ohittaa URL:a muokkaamalla käsiteltävän henkilön tunnistuksen. Tunnistus tarkoittaa tässä yhteydessä henkilötunnuksen vertaamista henkilön nimitietoihin tai syntymäaikaan. Tällöin olisi mahdollista, että käyttäjä muokkaa vahingossa väärän henkilön tietoja. Samaten mahdollisuus ohittaa sovellusten käytöstä perittävä laskutus nähtiin ongelmana. Molemmat potentiaaliset ongelmat olisi ollut mahdollista ratkaista sovelluksissa miettimällä sovellusten käyttötapoja ja kuinka tietoja haetaan taustajärjestelmistä, mutta tähän ei haluttu lähteä. Tämän vuoksi selvitettiin eri mahdollisuuksia tietojenvälitykseen. Selvityksessä tunnistettiin seuraavat tietojenvälitystavat: URL-parametrit, sessiotietokanta + URL-parametri, http-otsikkotiedot, session-/tiedonjako erillisellä kirjastolla ja portaali. Vaihtoehtojen hyviä ja huonoja puolia on listattu liitteessä 6. *Tietojenvälitystapojen vertailu*.

Vertailussa URL-parametrien käyttö tarjosi selkeästi eniten hyötyjä, mutta vain vähän potentiaalisia ongelmia. Siksi päädyttiin ehdottamaan seuraavaa. Välitettävät tiedot salattaisiin ja niistä muodostettaisiin yksi tiivistetty parametri, jota välitettäisiin näyttöjen ja sovellusten välillä. Tällöin käyttäjä ei pääse muokkaamaan URL:a ja voidaan asettaa URL:eille elinaika. URL-parametrien ja siten JSF:n näyttöparametrien käyttö hyväksyttiin lopulta tällaisena versiona.

Ratkaisussa päädyttiin myös tarjoamaan suosituksia uusittaville sovellusosille. Koska tiedot pitää hakea joka näytöille joka tapauksessa jostain, suositeltiin välitettävien tietojen käsittelyyn liittyvän erillisen sovellusmoduulin toteuttamista sovelluksissa, joka vastaisi tietojen hausta. Tämä moduuli hakisi tiedot sessiosta tai taustajärjestelmästä riippuen siitä, onko tietoja tarvittu aikaisemmin. Tällöin laskutukset syntyisivät aina, mutta sovelluslogiikkaa ei tarvittaisi enempää.

5.1.7 Näyttöparametrien salaaminen

Näyttöparametrien salauksen tavoitteena oli estää näyttöparametrien muokkaaminen käyttäjän toimesta. Salauksen tavoitteena ei ollut puuttua tietojen autentikointiin tai auktorisointiin. Tämän vuoksi salaustoiminnallisuus voitiin jättää melko suoraviivaiseksi ja käyttää symmetristä algoritmia käyttäjän toimialueen laajuisen SiteMinder-session tunnuksen toimiessa avaimena. Tämän vuoksi muodostetut linkit ovat voimassa vain kirjautuneelle käyttäjälle tämän yhden kirjautumisen ajan.

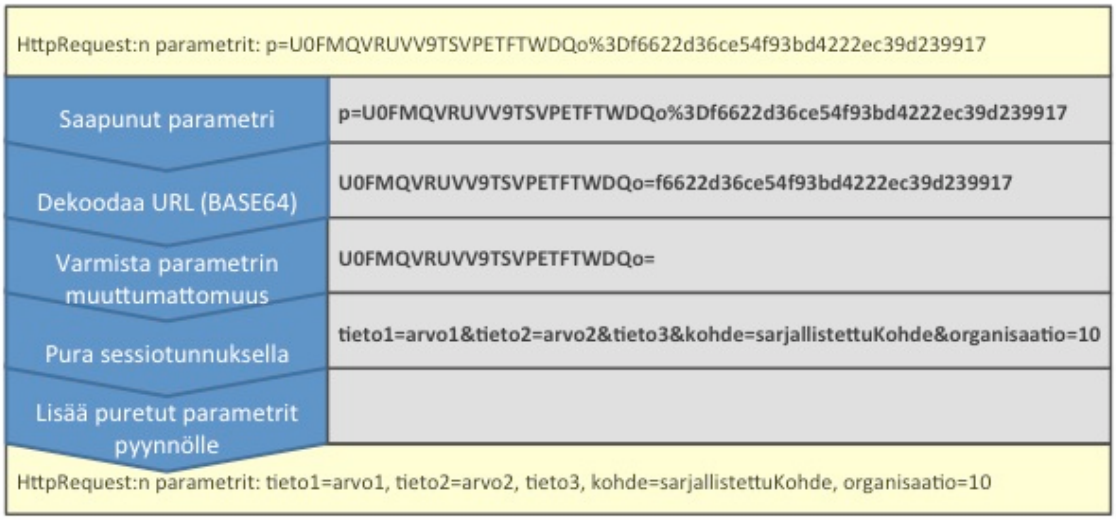
Näyttöparametrien salauksen vaiheet on esitetty kuvassa 30.

Kuva 30. Näyttöparametrien salauksen vaiheet

Näyttöparametrien salaus alkaa lisäämällä näyttökohtaiset lisätiedot, kuten käsiteltävä kohde ja kohdeorganisaatio, mikäli nämä on asetettu. Tämän jälkeen parametrilista salataan symmetrisellä algoritmilla, jonka avaimena käytetään käyttäjän SiteMinder session -tunnusta, joka on käytettävissä koko toimialueella käyttäjän session ajan. Salatut näyttöparametrit muunnetaan BASE64 koodatuksi merkkijonoksi, josta lasketaan MD5-tiiviste. Tiivisteen tarkoituksena on varmistaa parametrien muuttumattomuus. Tiiviste lisätään salattujen parametrien perään, jonka jälkeen parametri-merkkijono URL-koodataan UTF-8-merkistölle. Lopuksi salatut parametrit lisätään URL:n alkuperäisten parametrien tilalle avaimella ”p”.

Salaus toiminnallisuutta varten laajennettiin JSF:n navigaatiokäsittelijää (ConfigurableNavigationHandler), jossa pystyttiin käsittelemään suoraan JSF:n valmiiksi tuottamaa parametrilistaa ja korvaamaan se itse muodostetulla, salatulla parametrilla.

Näyttöparametrien purkamisen vaiheet on esitetty kuvassa 31.



Kuva 31. Näyttöparametrien purkaminen

Näyttöparametrien purkaminen alkaa dekoodaamalla saapunut parametri ”p” BASE64 muotoon. Tämän jälkeen parametrista poistetaan lähetettäessä muodostettu tiiviste ja muodostetaan salatusta osasta uusi tiiviste. Tiivisteitä verrataan keskenään, jotta pystytään varmistumaan, ettei parametria ole muokattu. Tämän jälkeen parametrit puretaan symmetrisellä algoritmilla käyttäen käyttäjän SiteMinder session tunnusta. Puretut parametrit käydään läpi ja lisätään pyynnölle (HttpServletRequest) salatun parametrin ”p” tilalle.

Purkamistoiminnallisuutta varten jouduttiin käyttämään JSF:n vaihekuuntelijaa heti näkymän palautuksen jälkeen. Koska pyynnön (HttpServletRequest) parametreja ei pysty lisäämään eikä poistamaan, jouduttiin laajentamaan sekä pyyntöä (HttpServletRequestWrapper), että pyynnön sisältävän kontekstin tuottavaa tehdasta (ExternalContextFactory). Laajennettuun pyyntötoteutukseen lisättiin toiminnallisuus parametrien lisäämiseksi ja poistamiseksi. Laajennettuun kontekstitehtaaseen lisättiin alkuperäisten pyyntöjen ja vastausten korvaaminen laajennetuilla toteutuksilla.

5.1.8 Navigointimalli

JSF tukee kahta eri tapaa määritellä navigointitapauksia. Ne ovat eksplisiittinen ja implisiittinen.

Eksplisiittisessä navigoinnissa etsitään toimintotunnusta vastaavaa navigointitapausta JSF:n konfiguraatiotiedostosta. Toimintotunnuksen ei täydy viitata mihinkään tiettyyn näyttöön, vaan se voi olla myös looginen tunnus, kuten ”ok” tai ”virhe”. Eksplisiittinen navigointitapa vaatii näyttöparametrien eriyttämisen toimintotunnuksesta. Näyttöparametrit voidaan määritellä esimerkiksi lähtösivulla omina parametreinaan tai navigaatiotapausten määrittelyn yhteydessä. Mikäli näyttöparametrit annetaan osana toimintotunnusta, eksplisiittinen navigointi epäonnistuu. Eksplisiittinen navigointitapa mahdollistaa myös oletusnavigointitapausten määrittelyn, jolloin esimerkiksi kaikilta sivuilta voidaan ohjata virhesivulle antamalla toiminnontunnus ”virhe”.

Implisiittistä navigointia käytetään, kun JSF:n konfiguraatiotiedostosta ei löydy painikkeeseen liitettyä navigointitapausta. Tässä tapauksessa luodaan virtuaalinen eli implisiittinen navigointitapaus ajonaikana, mikäli toimintoa vastaava sivu löytyy. Monasti implisiittinen navigointi liittyy näkymäparametrien käyttämiseen. Implisiittisessä navigointimallissa toiminnon tunnuksen tulee täsmätä haluttuun kohdesivuun, esimerkiksi toiminnolla ”sivu?parametri1=arvo1” JSF yrittää etsiä sivua ”sivu.xhtml” (pääte riippuu JSF:n konfiguraatiosta). Periaatteena implisiittisessä navigoinnissa on, että kaikki tarvittava tieto on annettava toimintotunnuksen yhteydessä, koska JSF:n konfiguraatiotiedostoja ei käytetä. Tämä estää myös loogisten toimintotunnusten käytön.

Näistä navigointitavoista päädyttiin suosittelemaan puhtaasti eksplisiittistä navigointitapaa. Eksplisiittisessä navigointitavassa kaikki näyttösiirtymät on dokumentoitu keskitettyyn paikkaan, jolloin navigointitapausten ylläpito on huomattavasti helpompaa isojen sovellusten tapauksessa, kuin Java-luokkien lähdekoodeista etsimällä. Toiminnoille suositellaan käytettäväksi loogisia tunnuksia, mutta sovelluskehittäjät saavat käyttää myös näyttöjen tunnuksia näin halutessaan.

PRG-suunnittelumallin edellyttämä uudelleenohjaus voidaan määritellä yksittäisten navigointitapausten yhteydessä, mutta globaalisti sitä ei saa päälle. Tämän vuoksi laajennettiin JSF:n navigointikäsittelijää (ConfigurableNavigationHandler), jossa pakotettiin

uudelleenohjaus päälle. Laajennuksessa etsitään täsmäävä navigaatiotapaus ja muokataan sen osoitetta sisältämään uudelleenohjauksen parametri.

Näyttöparametrit linjattiin konfiguroitaviksi navigaatiotapauksiin. Oletuksena sovelluksen omat näyttöparametrit eivät ole käytössä, mutta ne voidaan aktivoida navigaatiotapauskohtaisesti.

Eksplisiittinen navigointitapa paisuttaa JSF:n konfiguraatiotiedoston ison sovelluksen yhteydessä täysin ylläpitämättömään kuntoon. Tämä estettiin ohjeistamalla sovelluskehittäjiä tuottamaan kullekin käyttötapaukselle oma konfiguraatiotiedostonsa. Alustan JSF-pääkonfiguraatiossa asetetaan yleiset navigaatiotapaukset, kuten virhe- ja ohjenäytöille siirtymiset. Sovelluskonfiguraatiossa (web.xml) esitellään kaikki sovelluksen ja alustan JSF-konfiguraatiotiedostot alkaen alustan pääkonfiguraatiosta.

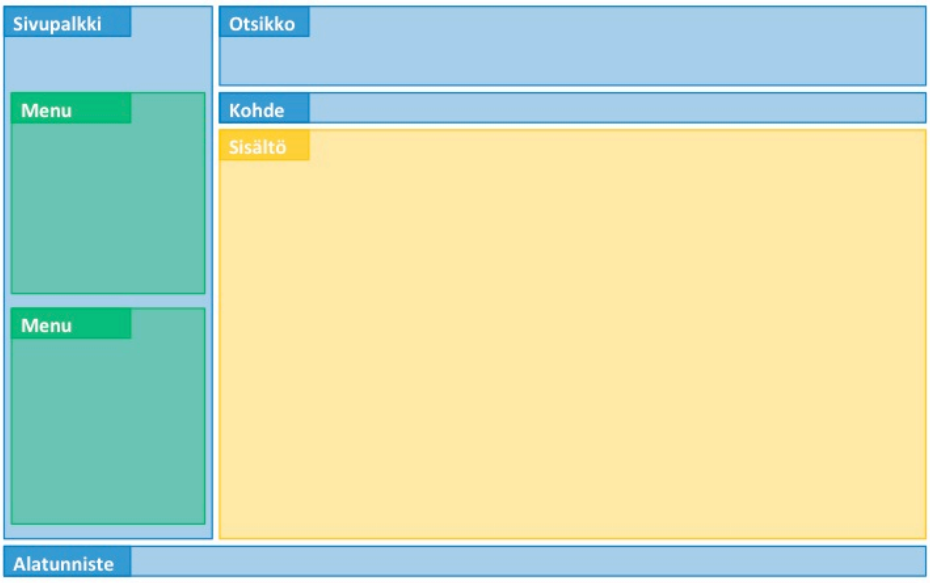
5.1.9 Näyttöpohjat

Näyttöpohjien tuottaminen on yksinkertaista käyttämällä JSF:n komposiittifaceletteja. Komposiittifaceleteille voidaan välittää käytettäessä parametrien arvoja ja määritellä/ylikirjoittaa näyttöpohjan nimettyjä osuuksia.

Projektissa ilmeni ongelmia, kun arkkitehtuuriosuudet haluttiin jaella valmiina paketteina. JSF 2.0 ei kuitenkaan tukenut EE5-ympäristössä näyttöpohjien lataamista suoraan JAR-pakettien sisältä. Tämän vuoksi projektissa jouduttiin tuottamaan oma JSF:n ExternalContext-luokka resurssien lataamiseksi. JSF-standardissa on määritelty lähes kaikki komponentit helposti korvattaviksi omilla toteutuksilla, joten oman laajennuksen tuottaminen oli suoraviivaista JSF:n näkökulmasta. JSF-toteutus osaa ladata osan tiedostoista suoraan JAR-paketista, mutta toisia taasen ei. JSF:ssä komponentteja laajennetaan yleisesti käärimällä oletustoiminnallisuus ja yliajamalla vain tarpeelliset osiot. JSF tarjoaa tätä varten omat kääreluokkansa, joista sovelluskehittäjät perivät omat toteutuksensa. Sovelluslaajennus toteutettiin tällä tavoin ja käytettiin itse tuotettua toiminnallisuutta vasta, mikäli JSF:n oletustoiminnallisuus ei kyennyt löytämään resurssia.

Perusnäytön näyttöpohjien rakenne on esitetty kuvassa 32. Sivupalkki, otsikko, alatunniste ja kohde muodostetaan alustan toimesta. Menu muodostetaan alustan ja sovel-

luksen yhteistyönä. Sovelluskehittäjien tulee tuottaa vain sisältö sekä konfiguroida menu.



Kuva 32. Perusnäytön näyttöpohjien rakenne

5.1.10 Teematuki

Asiakkaan käyttöliittymästandardi määritteli, että arkkitehtuurin olisi tuettava kahta erillistä teemaa. Toiminnallisuus liittyi asiakkaan ja toisen organisaation vanhoihin vastuualueisiin ja sovellusten tulisi näyttää eri organisaatiolta erikseen määritellyissä tapauksissa.

Teematuki rakennettiin yksinkertaisesti käyttämällä JSF:n resurssikirjastotoiminnallisuutta. Toiminnallisuus mahdollistaa resurssien, esim. kuvien ja tyylitiedostojen, jäsentelyn ja ryhmittelyn. Toiminnallisuutta varten luotiin sivupohjiin parametri, jonka avulla vaihtoehtoinen resurssikirjasto otettiin käyttöön.

5.2 Komponenttikirjasto

Komponenttikirjaston komponentit sisältävät valmiiksi asiakkaan käyttöliittymästandardin mukaiset tyylit ja asemoinnit sekä tietotyyppien tuen. Lisäksi kmponenttikirjasto sisältää tukikomponentteja mm. asemointia ja asetuksia varten.

Komponenttikirjaston tarkoituksena on

- helpottaa sovelluskehitystä, kun kehittäjät voivat keskittyä toiminnallisuuteen tyylittelyn ja ulkoasun sijasta
- auttaa erottamaan näytön tietosisältö ulkoasusta ja asemoinnista, joka mahdollistaa arkkitehtuuria käyttävien sovellusten päivittämisen käyttämään uusia käyttöliittymästandardin versioita tai tyylejä keskitetysti pienellä vaivalla
- tarjota yleinen toiminnallisuus, esim. Ajax-tapahtumatuki, asiakaspäänvalidaatio ja auktorisointi
- tarjota asiakkaan käyttöliittymästandardissa määritellyt ei-standardit toiminnallisuudet, esim. välilehdet.

Komponenttikirjaston toteutuksen lähtökohtana oli

- rakentaa komponenttihierarkia, jolloin yhteisen, uuden toiminnallisuuden lisääminen tietyn tyyppisille tai kaikille komponenteille onnistuisi keskitetysti
- olla riippumaton kolmannen osapuolen JavaScript-kirjastoista ja -moduuleista, kun JSF sisältää naiivin Ajax-tuen
- olla JSF:n osalta toteutusriippumaton, jolloin voidaan käyttää eri JSF-toteutuksia
- käyttää Javaa, missä mahdollista, jolloin Komponenttikirjaston ylläpito ja refaktorointi on helpompaa työkalutuen vuoksi
- komponenttien dokumentointi tulisi keskittää komponenttien toteutukseen ja dokumentaatio sekä tarvittavat resurssit, mm. tagikirjasto, tulisi tuottaa näiden perusteella käyttäen generaattoreita.

Projektin alussa mietittiin valmiin komponenttikirjaston käyttämistä sellaisenaan tai toteutuksen pohjana. Mikäli ulkopuolista komponenttikirjastoa olisi käytetty sellaisenaan, ei komponenttikirjasto olisi sisältänyt tukea asiakkaan käyttöliittymästandardin tyyleille ja asemoinneille. Mikäli taasen kirjastoa olisi käytetty toteutuksen pohjana, tulisi kirjasto kokonaisuudessaan asiakkaan ylläpidettäväksi, koska kirjaston muutoksia olisi hankala tuoda mukaan muokattuun kirjastoon. Joka tapauksessa tämä toisi uuden riippuvuuden, ja komponenttikirjastojen keskinäinen kilpailu ja siten oletettu elinkaari on lyhyempi, kuin itse standardin toteutuksilla.

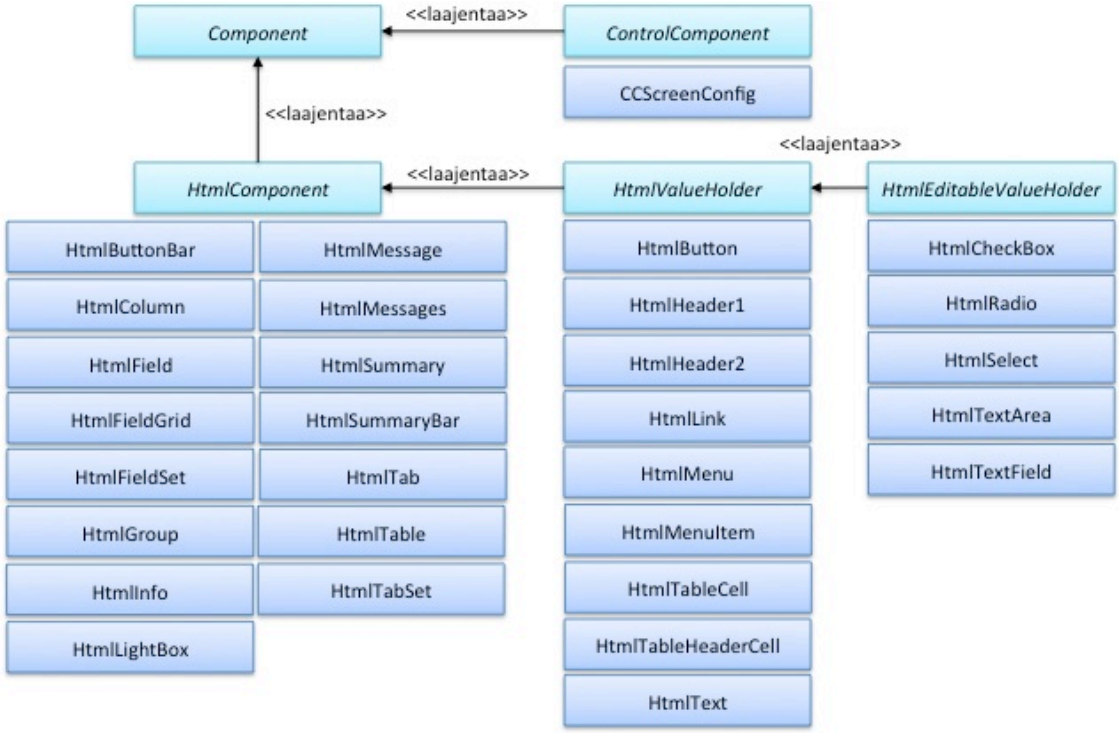
Komponenttikirjastoa yritettiin toteuttaa usealla eri tavalla. Alun perin kirjasto oli tarkoitus rakentaa käyttäen JSF 2.0:n uusia komposiittikomponentteja. Komposiittikomponentit ovat helppoja tuottaa, ja ne toimivatkin monessa mielessä hyvin, mutta osaa

toiminnallisuuksista, mm. kuuntelijoita, ei saatu toimimaan niiden kanssa kunnolla. Samaten komposiittikomponentille näytöllä annettu sisältö saatettiin tuottaa vain kerran per komposiittikomponentin toteutus. Tämä esti annetuista parametreista riippuvien vaihtoehtoisten toteutusten käyttämisen komponentin sisällä. Tämän jälkeen yritettiin käyttää hybridimallia, jossa komposiittikomponenttien taakse sidottiin Java-toteutus, jonne sijoitettiin logiikkaa. Tällä päästiin eteenpäin, mutta vaikeampien, mm. taulukko, komponenttien toteutuksessa ilmeni hankaluutta. Lopulta komponenttikirjasto päädyttiin toteuttamaan puhtaana Java-toteutuksena tuottaen komponenttimallit ja erilliset renderoijat.

Komponenttikirjasto osoittautui odotettua isotöisemmäksi, koska komponentit jouduttiin rakentamaan alusta alkaen. JSF-standardia ei määritellyt komponentteja tai renderoijia, joita voisi laajentaa omassa toteutuksessaan, joten kaikkien standardikomponenttien toteutukset ja renderoijat olivat toteutusspesifisiä. Tämän vuoksi komponentit jouduttiin rakentamaan täysin itse. Jopa tilanhallinta piti toteuttaa itse. Toteutuksessa otettiin vahvasti mallia Oraclen Mojarra -referenssitoteutuksesta, jonka toteutus oli huomattavasti Apachen MyFaces-version toteutusta parempitasoinen lähdekoodin selkeyden perusteella. Suurimmat vaikeudet toteutuksessa liittyivät tilanhallintaan ja JavaScript-toiminnallisuuksiin.

5.2.1 Komponentit

Komponenttikirjaston komponentit on esitetty komponenttihierarkian mukaisesti kuvassa 33. Turkoosilla värillä on kuvattu abstraktit pääluokat, jotka sisältävät komponenttikirjaston yhteisen toiminnallisuuden. Sinisellä värillä on kuvattu toteutuskomponentit.



Kuva 33. Komponenttikirjaston komponentit

Component on kaikkien komponenttien abstraktipääluokka, joka sisältää kaikille komponenteille yhteisen toiminnallisuuden sekä mm. attribuuttien, facettien ja tilanhallintaan liittyvät aputoiminnallisuudet.

ControlComponent on abstrakti pääluokka kontrollikomponenteille, jotka eivät tuota HTML-sisältöä, vaan joita käytetään esim. asetusten hallintaan. Koska kontrollikomponentit eivät tuota sisältöä, eivät ne tarvitse myöskään renderoijia. Kontrollikomponentteja käytetään esimerkiksi CCScreenConfigin tapauksessa yhteisten näyttöpohjien asetusten muokkaamiseen.

HtmlComponent on abstrakti pääluokka kaikille HTML-sisältöä tuottaville komponenteille. Komponentti sisältää tuen kaikille HTML-attribuuteille ja tapahtumille sekä komponenttitason profiloinnille.

HtmlValueHolder on abstrakti pääluokka kaikille komponenteille, jotka sisältävät näytettävän arvon. Komponentti sisältää tuen asiakkaan käyttöliittymästandardissa määritellyille yleisille tietotyypeille JSF:n konvertterien avulla ja arvon tallentamiseen liittyvän tilanhallintatoiminnallisuuden.

HtmlEditableValueHolder on abstrakti pääluokka kaikille komponenteille, jotka sisältävät muokattavan arvon. Luokka laajentaa HtmlValueHolderin yleisten tietotyyppien tukea validaattorien osalta.

Komponenttien alustus

Konstruktoria kutsuttaessa komponentin arvoja ei ole vielä asetettu, joten sitä ei voida käyttää esimerkiksi annettujen arvojen validointiin tai alustamiseen oletusarvoilla. Monet Java-komponentit tukevat alustusta esimerkiksi @PostConstruct-annotaatiolla, mutta JSF:n tapauksessa komponentit eivät tätä tue, eikä mitään vastaavaa mekanismiakaan ole. Komponenttikirjaston tapauksessa tämä ratkaistiin lisäämällä pääkomponentille järjestelmätapahtumienkuuntelija (ComponentSystemEventListener), joka kutsui komponentissa määriteltyä alustusmetodia, kun komponentti lisättiin näkymään. Näkymään lisättäessä komponentilla on kaikki annetut arvot käytettävissään, kuten myös JSF:n konteksti. Aliluokat voivat yliajaa tämän alustusmetodin lisätäkseen oman alustustoiminnallisuutensa. Ratkaisun erityispiirteenä oli, että metodia kutsuttiin myös, kun komponenttia siirrettiin JSF:n muodostamassa näkymäpuussa paikasta toiseen, jolloin toteutuksissa piti varautua useaan kutsuun per pyynnönkäsittely.

Auktorisointi

JSF-komponenteilla on totuusarvotyyppinen kenttä, joka kertoo renderoidaanko komponentti. Auktorisointia varten HTML-komponenteille lisättiin uusi kenttä auktorisointitunnuksen asettamista varten. Auktorisointitoteutus lisättiin laajentamalla komponentin renderointitiedon palauttavaa metodia. Renderoinnin yhteydessä tutkitaan, onko auktorisointitunnus asetettu ja mikäli on, kutsutaan alustan auktorisointitoteutusta. Mikäli käyttäjällä ei ole oikeuksia nähdä komponentin sisältöä, estetään komponentin renderointi kokonaan.

Yleiset tietotyypit

Yleiset tietotyypit toteutettiin, jotta kaikilla näytöillä tietoja käsiteltäisiin yhtenäisesti ja eri projektien ei tarvitsisi toteuttaa samoja konverttereita ja validaattoreita. Projektin aikana tuki lisättiin seuraaville tietotyypeille: päivämäärä, rahasumma, henkilötunnus, Y-tunnus, vakuutusnumero ja prosenttiluku.

Yleisten tietotyyppien tuki rakennettiin kahteen eri paikkaan. HtmlValueHolder-luokka sisältää näytettävän arvon, jonka vuoksi luokka tarvitsee muunninluokan tietojen muuntamiseksi näytöllä näytettäväksi ja takaisin Java-olioksi. Tietotyypin asettamista varten lisättiin uusi attribuutti, jonka arvona käytettiin tätä tarkoitusta varten luotua enumeraatiota. HtmlEditableValueHolder-luokka sisältää muokattavan arvon, jonka vuoksi luokka tarvitsee lisäksi validaattoriluokkia syötettyjen arvojen validoimiseksi. Mikäli sovelluskehittäjä asettaa komponentille tietotyypin, Komponenttikirjaston luokat osaavat ottaa tarvittavat muunnin- ja validaattoritoteutukset käyttöön automaattisesti, jolloin sovelluskehittäjän ei tarvitse esitellä niitä joka näytöllä erikseen.

Asiakaspään validaatio

Asiakaspään validaatio toteutettiin käyttäen JSF 2.0:ssa esiteltyä Ajax-toiminnallisuutta. Toteutus käyttää sovelluskehittäjien ja komponenttikirjaston lisäämiä muunnin- ja validointiluokkia. Tällä tavoin asiakaspäässä voitiin käyttää samaa validaatiototeutusta, kuin palvelinpäässä. Lisäksi Komponenttikirjastoa varten ei tarvinnut ottaa käyttöön kolmannen osapuolen JavaScript-kirjastoja.

Toiminnallisuus on toteutettu HtmlEditableValueHolder-luokassa ja sen renderoijassa HtmlEditableValueHolderRenderer. Renderoija lisää tarvittaessa komponentille ajonaikaisesti JSF:n viestikomponentin lapseksi, joka sisältää käyttäjälle näytettävän virheviestin.

5.2.2 Renderoijat

JSF:n komponentit muodostavat käyttäjälle näytettävän HTML-sisällön käyttäen konfiguroitua renderoijaa tai yksinkertaisissa tapauksissa itse. Renderoijia voidaan käyttää muodostamaan tietosisältöä useille erilaisille komponenteille. Komponenttikirjastossa kaikki sisältö tuotetaan käyttäen renderoijia. Renderoijille luotiin pääluokka HtmlRenderer, joka toteuttaa JSF:n Renderer-rajapinnan ja tarjoaa yleiset tyyppi- ja tilatarkistukset. Pääluokka tarjoaa aliluokilleen oman tyypitetyn sovellusrajapinnan, eri vaiheiden yleisimmät tiedot sekä useita muita apumetodeja. Renderoijien sisäinen hierarkia muistuttaa hyvin pitkälti komponenttikirjaston sisältökomponenttien hierarkiaa.

5.2.3 Dokumentaatio

Komponenttikirjaston dokumentaatio toteutettiin omana käyttöliittymäsovelluksenaan. Sovellus sisältää kaikille komponenteille dokumentaation ja toimivia esimerkkejä, joita sovelluskehittäjä voi tutkia. Lisäksi dokumentaatio sisältää muutamia esimerkkisivuja, joilla esitellään laajemmin kirjaston ja alustan ominaisuuksia toiminnassa.

5.2.4 Eclipsen automaattitäydennys komponenttikirjaston komponenteille

Ilman automaattitäydennystä kehittäjät joutuvat muistamaan ulkoa tai katsomaan erikseen dokumentaatiosta joka ikisen komponentin nimen ja kirjoitusasun sekä attribuuttien nimet, tyypit ja mahdolliset arvot. Mikäli yksikin arvo on pielessä, sivun renderointi keskeytyy virheeseen. Tämä hidastaa kehitystä huomattavasti ja aiheuttaa turhautumista teknologiaa kohtaan. Automaattitäydennys tarjoa listaa komponenttikirjaston komponenteista kuvauksineen, lisää komponentin valittaessa sen pakolliset attribuutit sekä tarjoaa listaa mahdollisista lisättävistä attribuuteista kuvauksineen. Lisäksi automaattitäydennys tarjoaa validaation työstettäville Facelet-sivuille, mikä vähentää kehityksen aikaisten virheiden määrää ja vähentää turhautumista.

JSF:n komponentit tarvitsevat määrittelytiedoston, joka kokoaa komponenteista Facelet näytöillä käytettävän komponenttikirjaston. Tässä tiedostossa määritellään komponenttikirjaston nimi ja nimiavaruus. Tämän lisäksi omille komponenteille ainoita pakollisia tietoja ovat komponentin ja renderoijan nimet, muut tiedot ovat metatietoa komponenteista.

Eclipsen Java EE -version automaattitäydennys tarvitsee toimiakseen komponenttikirjaston määrittelytiedoston lisäksi vanhentuneen tagikirjaston määrittelytiedoston. Tiedoston sisällöksi riittää kirjaston nimi ja nimiavaruus, joiden täytyy täsmätä varsinaisessa kirjaston määrittelytiedostossa annettuja tietoja toimiakseen. Eclipsen automaattitäydennys käyttää komponenttikirjaston määrittelytiedostosta vain attribuuttien nimiä, tietoa pakollisuudesta ja kuvausta. Tämän vuoksi Komponenttikirjaston generaattori pyrkii sisällyttämään attribuuttien kuvauksiin mahdollisimman paljon lisätietoa, kuten attribuuttien tyypit, oletusarvot ja mahdolliset arvot enumeraatioiden tapauksessa.

Lopputuloksena automaattinen täydennys saatiin toimimaan verrattain hyvin omien komponenttien osalta.

6 Arvio projektin onnistumisesta

Projektin tavoitteena oli korvata asiakkaan suljetun sovelluskehyksen käyttöliittymäkerros standardipohjaisella ratkaisulla. Käyttöliittymäprojektin tuottamasta yhteisestä toiminnallisuudesta saatiin tuotettua asiakkaalle pohja uudeksi sovellusalustaksi, mitä ei projektien käynnistyessä edes uskallettu toivoa. Alusta viimeisteltiin kolmannen projektin toimesta, ja se valmistui samaan aikaan kahden muun projektin kanssa. Uusi sovellusalusta mahdollistaa asiakkaalle uusien sovellusten tuottamisen ja vanhojen päivittämisen käyttäen Java EE standardin mukaisia teknologioita. Uudella alustalla voidaan myöhemmin luopua kokonaan massiivisesta vanhasta alustasta, joka korvautuu standarditeknologioilla ja niiden päälle tehdyillä pienillä laajennoksilla. Tällöin asiakkaan ylläpitämän alustatoteutuksen määrä pienenee murto-osaan, kuten myös ylläpitokustannukset. Lisäksi standarditeknologiat ovat sovelluskehittäjille tuttuja ja niille löytyy valmista koulutusmateriaalia, kirjoja ja kursseja, jolloin asiakas voi luopua myös omasta koulutusmateriaalistaan ja koulutuksistaan, jotka ovat vieneet tähän asti useiden henkilöiden työpanoksen. Näiden perusteella projekti onnistui yli odotusten.

Projektin aikana kaikki ei kuitenkaan sujunut aina suunnitelmien mukaan. Käyttöliittymäprojektilaiset joutuivat esimerkiksi projektin alussa lähes kuukaudeksi pelastamaan toista projektia, mikä aiheutti koko loppuprojektin ajan ongelmia kiinnitettyjen aikataulujen vuoksi. Kiinteät aikataulut johtuivat esimerkiksi käyttöliittymä- ja SOAP-kanavien yhteisestä arkkitehtuurista ja riippuvuuksista muiden projektien kanssa. Jälkeenpäin ajatellen yhteisen arkkitehtuurin toteutukselle olisi pitänyt varata selkeämmin oma projektinsa ja määrittelynsä. Samaten kahden ison projektin päällekkäinen aloittaminen ei varmaan ollut kaikista paras ratkaisu. Projektit olisi voitu lomittaa vaikka siten, että yhteiset osat olisi saatu toteutettua ja toisen projektin alustava arkkitehtuurisuunnitelma valmiiksi ja toteutus käyntiin ennen toisen käynnistämistä.

Lisäksi pääosin aikataulusyistä johtuen ensimmäinen sovelluskonversio osoittautui hyvin ongelmalliseksi. Konversio käynnistettiin aikataulupaineiden vuoksi aivan liian aikaisessa vaiheessa ja sovelluskehittäjät joutuivat painimaan keskeneräisten ominaisuuksi-

en parissa. Myös konvertoitavasta sovelluksesta paljastui useita puutteita. Jälkeenpäin ajatellen konversiota olisi pitänyt lykätä, kunnes itse alusta olisi ollut edes suunnilleen kasassa.

Käyttöliittymäteknologian valinta osoittautui erittäin onnistuneeksi ja helposti laajennettavaksi käyttöliittymäarkkitehtuurin osalta. Käyttöliittymäarkkitehtuuri saatiin toteutettua muutamalla JSF:n julkisen sovellusrajapinnan laajennuksella toimivaksi kokonaisuudeksi.

Komponenttikirjaston toteutuksen yhteydessä esiintyi turhankin monia takaiskuja. JSF:n standardikomponenttien toteutusriippumattoman laajennusmahdollisuuden puuttuminen oli melkoinen pettymys. Paras lähestymistapa olisi jälkikäteen ajatellen ollut siirtyä suoraan tekemään komponentteja alusta alkaen itse.

Kokeiluissa uudella sovellusalustalla ja käyttöliittymäarkkitehtuurilla saatiin toteutettua valmiita näyttöjä murto-osalla vanhan sovellusalustan vaatimasta ajasta. Vanhalla sovellusalustalla perusnäytön tekeminen vaati haastattelujen mukaan 5-7 htp (henkilötyöpäivää) ensimmäisiä näyttöjään toteuttavalta kehittäjältä ja 3-5 htp kokeneemmalta kehittäjältä. Mikäli näytöllä oli monimutkaisempaa toiminnallisuutta, kuten järjestely, nousivat työmäärät parillä päivällä. Mikäli sivulle tuli lisätä profilointituki, nousivat työmäärät vielä 5-10 htp lisää. Uudella sovellusalustalla kokeiluissa kehittäjät saivat toteutettua ensimmäiset perusnäyttönsä alle päivässä ja seuraavat noin puolessa päivässä. Järjestelyt ynnä muut vastaavat esitettävän tiedon muokkaukset eivät nostaneet työmääriä merkittävästi. Profilointituen lisääminen vie alle puoli päivää, mikäli auktorisointitiedot on määritelty.

7 Yhteenveto

Projektin tavoitteena oli tuoda moderni sekä laajasti tunnettu ja tuettu käyttöliittymäteknologia/tuote asiakkaan vanhan teknologian rinnalle. Tällöin teknologia olisi toimittajille entuudestaan tuttua ja sille löytyisi ulkopuolista dokumentaatiota, oppaita ja koulutusta. Tunnetut teknologiat parantaisivat toimittajien toimitusvalmiutta, vähentäisivät työmääriä ja lisäisivät asiakkaan houkuttelevuutta sovelluskehittäjille.

Projekti käynnistyi syksyllä 2011 vaatimusmäärittelyllä, jonka työpajoissa kerättiin 157 vaatimusta ja 12 rajausta.

Teknologianvalinta käynnistyi kriteerien tunnistamisella. Hyväksytylle kriteerilistalle laadittiin painokertoimet ja siirryttiin vertailtavien teknologioiden tunnistamiseen. Vertailuun päätyivät seuraavat teknologiat: JSF (2.0), Spring MVC (3.0.7), Wicket (1.5.2), GWT (2.4), Tapestry (5.2.6), Grails (1.3.7), Vaadin (6.7.1) ja Struts (2.2.3.1). Vertailumatriisin perusteella esiin nousivat JSF 2.0 ja Struts 2, joista PoC-testaukseen valittiin vain Java EE -standardin mukainen JSF 2.0. JSF 2.0 pärjäsi PoC-testauksessa hyvin ja se hyväksyttiin PoC-sovelluksen demotilaisuudessa yksimielisesti käytettäväksi käyttöliittymäteknologiaksi.

Toteutus alkoi vuoden 2012 alussa, jolloin käynnistettiin käyttöliittymä- ja SOAP-kanavien yhteisen arkkitehtuurin toteutus. Yhteinen arkkitehtuuri toteutettiin Java EE6 standardin CDI-teknologian ympärille. Koska yhteinen arkkitehtuuri muodosti hyvän pohjan standardipohjaiselle sovellusalustalle, käynnistettiin sen viimeistelemiseksi asiakkaan uudeksi sovellusalustaksi oma projektinsa.

Käyttöliittymäosuuden toteutus alkoi keväällä 2012. Käyttöliittymäosuus koostuu kahdesta osasta: käyttöliittymäarkkitehtuuri ja Komponenttikirjasto. Käyttöliittymäarkkitehtuuri koostuu JSF 2.0:sta ja projektin aikana sen päälle tuotetuista laajennoksista. Laajennoksien pääasiallisena tarkoituksena on sitoa JSF 2.0:n pyynnönkäsittelyn elinkaaren vaiheet yhteisen sovellusalustan palveluihin sekä tarjota puuttuvat liiketoiminnalliset vaatimukset. Komponenttikirjaston komponentit sisältävät valmiiksi asiakkaan käyttöliittymästandardin mukaiset tyylit ja asemoinnit sekä tietotyyppien tuen. Lisäksi Komponenttikirjasto sisältää tukikomponentteja mm. asemointia ja asetuksia varten.

Projekti valmistui alkuvuodesta 2013, jolloin projektin pääloppututuotteet olivat sovellusprojektien käytettävissä.

Käyttöliittymäprojektin tuottamasta yhteisestä toiminnallisuudesta saatiin tuotettua asiakkaalle pohja uudeksi sovellusalustaksi, mitä ei projektien käynnistyessä edes uskallettu toivoa. Uusi sovellusalusta mahdollistaa asiakkaalle uusien sovellusten tuottamisen ja vanhojen päivittämisen käyttäen Java EE -standardin mukaisia teknologioita. Uudella alustalla voidaan myöhemmin luopua kokonaan massiivisesta vanhasta alustas-

ta, joka korvautuu standarditeknologioilla ja niiden päälle tehdyillä pienillä laajennoksilla. Tällöin asiakkaan ylläpitämän alustatoteutuksen määrä pienenee murto-osaan, kuten myös ylläpitokustannukset. Lisäksi standarditeknologiat ovat sovelluskehittäjille tuttuja ja niille löytyy valmista koulutusmateriaalia, kirjoja ja kursseja, jolloin asiakas voi luopua myös omasta koulutusmateriaalistaan ja koulutuksistaan, jotka ovat vieneet tähän asti useiden henkilöiden työpanoksen. Näiden perusteella projekti onnistui yli odotusten.

Lähteet

1. Bradner, S. 1997. RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels. Verkkodokumentti. <http://www.ietf.org/rfc/rfc2119.txt>. Luettu 16.8.2011.

2. Raible, M. 2010. Comparing JVM Web Frameworks. Verkkodokumentti. <http://www.slideshare.net/mraible/comparing-jvm-web-frameworks>. Luettu 1.11.2011.

3. Boe, B. 2011. Using StackOverflow's API to Find the Top Web Frameworks. Verkkodokumentti. <http://www.bryceboe.com/2011/02/21/using-stackoverflows-api-to-find-the-top-web-frameworks/>. Luettu 30.10.2011.

4. LinkedIn advanced people search. 2011. Verkkodokumentti. LinkedIn Corporation. <http://www.linkedin.com>. Luettu 6.11.2011.

5. Oracle Mojarra JavaServer Faces community page. 2011. Verkkodokumentti. Oracle Corporation. <https://javaserverfaces.java.net>. Luettu 30.10.2011.

6. Apache MyFaces home page. 2011. Verkkodokumentti. Apache Software Foundation. <http://myfaces.apache.org>. Luettu 30.10.2011.

7. Spring Framework home page. 2011. Verkkodokumentti. SpringSource Inc. <http://www.springsource.org>. Luettu 30.10.2011.

8. Apache Wicket home page. 2011. Verkkodokumentti. Apache Software Foundation. <http://wicket.apache.org>. Luettu 30.10.2011.

9. GWT home page. 2011. Verkkodokumentti. Google Inc. <http://www.gwtproject.org>. Luettu 30.10.2011.

10. Apache Tapestry home page. 2011. Verkkodokumentti. Apache Software Foundation. <http://tapestry.apache.org>. Luettu 30.10.2011.

11. Grails home page. 2011. Verkkodokumentti. SpringSource Inc. <http://grails.org>. Luettu 30.10.2011.

12. Vaadin home page. 2011. Verkkodokumentti. Vaadin Ltd. <https://vaadin.com/home>. Luettu 30.10.2011.

13. Apache Tapestry home page. 2011. Apache Software Foundation. Verkkodokumentti. <http://struts.apache.org/development/2.x/>. Luettu 30.10.2011.

14. Job Trends. 2011.. Verkkodokumentti. Indeed.com. <http://www.indeed.com/jobtrends>. Luettu 9.11.2011.

15. Amazon book search. 2011. Verkkodokumentti. Amazon.com Inc. <http://www.amazon.com>. Luettu 30.10.2011.

16. Apache Struts. 2011. Verkkodokumentti. Wikimedia Foundation Inc. <http://en.wikipedia.org/wiki/Apache_Struts>. Luettu 1.11.2011.

17. StackOverflow question and answer site. 2011. Verkkodokumentti. Stack Exchange Inc. <http://stackoverflow.com>. Luettu 8.11.2011.

18. Google trends. 2011. Verkkodokumentti. Google Inc. <http://www.google.com/trends/>. Luettu 9.11.2011.

19. Borges, B. 2011. Top 10 reasons why I don't like JSF. Verkkodokumentti. <http://blog.brunoborges.com.br/2010/12/top-10-reasons-why-i-dont-like-jsf.html>. Luettu 31.10.2011.

20. Gosling, J. 2010. James Gosling on Apple, Apache, Google, Oracle and the Future of Java. Verkkovideo. <http://www.youtube.com/watch?v=9ei-rbULWoA>. Katsottu 31.10.2011.

21. Google Groups: James Gosling on Apple, Apache, Google, Oracle and the Future of Java. 2011. Verkkodokumentti. Google Inc. <https://groups.google.com/forum/#!topic/javaposse/Q8K4EkfXxpM>. Luettu 31.10.2011.

22. Burns, E. 2012. POST-REDIRECT-GET and JSF 2.0. Verkkodokumentti. <https://blogs.oracle.com/enterprisetechtips/entry/post_redirect_get_and_jsf>. Luettu 13.2.2012.

Rajaukset

Taulukko 5. Rajaus 1: Järjestelmä täytyy toteuttaa Java-ohjelmointikielellä

Kuvaus	Järjestelmä täytyy toteuttaa Java-ohjelmointikielellä
Seuraukset	Vain Java-pohjaiset ratkaisut otetaan mukaan teknologiavertailuun.
Testaus	Ei testata.
Lähde	Arkkitehtuuritoimisto

Taulukko 6. Rajaus 2: Kaikkein järjestelmän rakentamiseen käytettävien komponenttien ja kirjastojen täytyy olla asiakkaan erikseen hyväksymiä

Kuvaus	Kaikkien järjestelmän rakentamiseen käytettävien komponenttien, kirjastojen ja työkalujen lisenssien täytyy olla asiakkaan lisenssikäytäntöjen mukaisia. Kaikkein järjestelmän rakentamiseen käytettävien komponenttien ja kirjastojen täytyy olla asiakkaan erikseen hyväksymiä.
Seuraukset	Kaikki komponentit, kirjastot ja työkalut ovat asiakkaan hyväksymiä ja heidän lisenssikäytäntöjensä mukaisia.
Testaus	Tutkitaan ja varmistetaan, että koostamis-, testaus- ja ajoympäristön luokkapolut eivät sisällä hyväksymättömiä kirjastoja. Työkalut joudutaan tutkimaan erikseen.
Lähde	Arkkitehtuuritoimisto, työpaja 2

Taulukko 7. Rajaus 3: Järjestelmän täytyy tukea kaikkia asiakkaan tukemia selaimia

Kuvaus	Järjestelmän, mukaan lukien sen alikomponentit, täytyy tukea kaikkia asiakkaan tukemia selaimia. Vaatimusmäärittelyä laadittaessa näitä olivat: Firefox 6, IE 7, IE 8 ja IE 9.
Seuraukset	Loppukäyttäjät voivat käyttää järjestelmän päälle rakennettavia sovelluksia kaikilla asiakkaan tukemilla selaimilla.
Testaus	Testataan manuaalisesti mm. asemoinnit ja JavaScript-toiminnallisuus.
Lähde	Arkkitehtuuritoimisto, käyttöliittymästandardi, työpaja 2

Taulukko 8. Rajaus 4: Kaikkien järjestelmän rakentamiseen käytettävien komponenttien täytyy olla yhteensopivia asiakkaan sovelluspalvelimien kanssa

Kuvaus	Kaikkien järjestelmän rakentamiseen käytettävien komponenttien täytyy olla yhteensopivia asiakkaan sovelluspalvelimien kanssa.
Seuraukset	Järjestelmää käyttävät sovellukset voidaan asentaa asiakkaan sovelluspalvelimille.
Testaus	Rakennetaan tuotannonkaltainen systeemitestiympäristö, jossa järjestelmän eri komponentteja voidaan testata yhdessä ja erikseen.
Lähde	Arkkitehtuuritoimisto, työpaja 2

Taulukko 9. Rajaus 5: Järjestelmän olisi hyvä pohjautua avoimenlähdekoodin ratkaisuihin

Kuvaus	Järjestelmän olisi hyvä pohjautua avoimenlähdekoodin ratkaisuihin. Kaupallinen tuki ei ole välttämättömyys, mutta toivottavaa.
Seuraukset	Lähdekoodit saatavilla ongelmanselvittelyä varten. Ei lisenssikuluja. Mahdollistaa järjestelmän myymisen/tarjoamisen palveluna. Mikäli komponentin tuki loppuu, voidaan komponentista tehdä oma versio - mikäli komponenttia ei ehditä korvata.
Testaus	Ei testata.
Lähde	Työpaja 1

Taulukko 10. Rajaus 6: Järjestelmän täytyy käyttää käyttöliittymästandardin määrittelemiä web-standardeja

Kuvaus	Järjestelmän täytyy käyttää käyttöliittymästandardin määrittelemiä web-standardeja. Näitä ovat: • XHTML 1.0 Strict (http://www.w3.org/TR/xhtml1/) • CSS 2.1 (http://www.w3.org/TR/CSS/) HTML5 ja CSS3 -teknologioita ei saa käyttää ennen kuin ne ovat laajasti käytettyjä ja lisätty käyttöliittymästandardiin.
Seuraukset	Toteutetut sovellukset toimivat todennäköisesti kaikilla tuetuilla selaimilla.

Testaus	Skannataan sivupohjat, mallitiedostot ja tyylitiedostot määriteltyjen standardien ulkopuolisien elementtien, attribuuttien ja määrittelyiden varalta. Tarkistetaan, että toteutetut näytöt käyttävät oikeita XHTML- ja CSS -määrittelyitä. Validoidaan toteutetut näytöt määriteltyjä standardeja vasten, esim. W3C:n työkaluilla tai palveluilla.
Lähde	Käyttöliittymästandardi

Taulukko 11. Rajaus 7: Järjestelmän täytyy käyttää käyttöliittymästandardin määrittelemää JavaScript versiota

Kuvaus	Järjestelmän täytyy käyttää käyttöliittymästandardin määrittelemää JavaScript versiota. Käytettävä versio on JavaScript 1.5 (ECMA-262 Edition 3). Asiakas on saanut palautetta omien asiakkaidensa tietoturvavastaavilta liittyen JavaScriptin käyttöön. He näkevät JavaScriptin mahdollisena tietoturvauhkana ja toivoisivat, että järjestelmää olisi mahdollista käyttää ilman JavaScriptiä.
Seuraukset	Toteutetut JavaScript-toiminnallisuudet toimivat todennäköisesti kaikilla tuetuilla selaimilla.
Testaus	Testataan osana rajausta kolme.
Lähde	Käyttöliittymästandardi, työpaja 1

Taulukko 12. Rajaus 8: Järjestelmän täytyy käyttää käyttöliittymästandardin mukaisesti UTF-8 -merkistöä

Kuvaus	Järjestelmän täytyy käyttää käyttöliittymästandardin mukaisesti UTF-8 -merkistöä.
Seuraukset	Vältytään merkistöongelmilta, kun koko järjestelmä - mukaan lukien tuotetut näytöt - käyttävät samaa merkistöä.
Testaus	Varmistetaan, että näyttöjen määrityksissä on asetettu UTF-8 -merkistö.
Lähde	Käyttöliittymästandardi

Taulukko 13. Rajaus 9: Järjestelmä täytyy rakentaa käyttäen asiakkaan koodausstandardeja ja muotoiluja

Kuvaus	Järjestelmä täytyy rakentaa käyttäen asiakkaan koodausstandardeja ja muotoiluja.
Seuraukset	Java-luokkien ylläpito ja vertailu on huomattavasti helpompaa yhtenäisillä koodausstandardeilla ja muotoiluilla.
Testaus	Osana koodikatselmointia.
Lähde	Työpaja 3

Taulukko 14. Rajaus 10: Järjestelmän täytyy käyttää käyttöliittymästandardin mukaisia kuvaformaatteja

Kuvaus	Järjestelmän täytyy käyttää käyttöliittymästandardin mukaisia kuvaformaatteja, joita ovat: PNG, JPEG ja GIF.
Seuraukset	Kuvat aukeavat ongelmitta tuetuilla selaimilla.
Testaus	Verifioidaan kuvien tiedostopäätteet.
Lähde	Käyttöliittymästandardi

Taulukko 15. Rajaus 11: Järjestelmän ei täydy tukea muita laitteita, kuin selaimia

Kuvaus	Järjestelmän ei täydy tukea muita laitteita, kuin selaimia. Mikäli tarve uudelle laitteelle nousee, voidaan sille lisätä tuki uutena kanavana.
Seuraukset	Järjestelmän arkkitehtuuri on yksinkertaisempi ja sen rakentaminen on nopeampaa.
Testaus	Ei testata.
Lähde	Arkkitehtuuritoimisto, työpaja 1, työpaja 2

Taulukko 16. Rajaus 12: Järjestelmän täytyy olla sovelluspalvelinneutraali

Kuvaus	Järjestelmän täytyy olla sovelluspalvelinneutraali. PoC- ja pilottitestaus tehdään käyttäen Apache Tomcat sovelluspalvelimia. Systeemitestivaiheesta eteenpäin käytetään IBM:n WebSphere sovelluspalvelimia.

Seuraukset	Järjestelmä ei sido asiakasta nykyisiin sovelluspalvelimiinsa, vaan sovellukset voidaan asentaa muihinkin sovelluspalvelimiin kuin IBM WebSphere.
Testaus	Pilottitestauksessa käytetään Apache Tomcat sovelluspalvelimia ja systeemitestiympäristössä IBM WebSphere sovelluspalvelimia.
Lähde	Arkkitehtuuritoimisto, työpaja 1, työpaja 2

Matt Raiblen Web teknologian valintakriteerit

Kriteeri	Kriteerin tulkinta	Käyttö-kelpoisuus
Kehittäjän tuottavuus	Kuinka tuottava kehittäjä on. Kuinka usein sovellus tarvitsee esimerkiksi asentaa sovelluspalvelimelle kehityssyklin aikana, kuinka kauan aikaa kuluu jokaiseen asennukseen, onko markkinoilta saatavilla teknologiaa tukevia työkaluja jne.	Vaikea mitata
Kehittäjän havaintokyky	Kuinka hyvin kehittäjä ymmärtää, miten sovellus toimii ja miten osa-alueet vaikuttavat toisiinsa.	Vaikea mitata
Oppimiskynnys	Kuinka nopeasti kehittäjä oppii käyttämään teknologiaa?	Kyllä
Projektin tila	Kuinka aktiivinen projekti on? Esim. julkaisujen, sähköpostilistojen ja foorumeiden viestien lukumäärä edeltävänä vuonna.	Kyllä
Kehittäjien saatavuus	Kuinka paljon osaavia kehittäjiä on saatavilla kyseiselle teknologialle.	Kyllä
Työpaikkatrendit	Kuinka paljon työpaikkoja on tarjolla millekin teknologialle ja minkälainen on työpaikkojen trendi.	Kyllä
Mallituki	Tukeeko teknologia mallien käyttöä käyttöliittymän sivujen rakentamisessa?	Kyllä
Komponenttituki	Tukeeko teknologia komponenttien käyttöä käyttöliittymän sivujen rakentamisessa?	Kyllä
Ajax-tuki	Tukeeko teknologia Ajax-teknologiaa suoraan, lisäosin vai ei lainkaan.	Kyllä
Liitännäis-/laajennustuki	Tukeeko teknologia omien sovellusliitännöisten/-laajennosten tekemistä ja käyttämistä?	Kyllä
Skaalautuvuus	Kuinka hyvin teknologia skaalautuu ympäristön resurssien mukaan, esim. sovelluspalvelimien suorituskyvyn, muistikapasiteetin ja lukumäärän mukaan.	Kyllä
Testattavuus	Kuinka helppoa teknologialla tuotettuja käyttöliittymäsivuja on testata?	Kyllä
Lokalisaatiotuki	Tukeeko teknologia lokalisaatiota (i18n, l10n) kuinka hyvin?	Kyllä
Validaatiotuki	Sisältääkö teknologia tukea datan validoimiselle?	Kyllä
Usean kielen -tuki	Tukeeko teknologia usean ohjelmointikielen käyttöä?	Ei
Dokumentaation taso	Kuinka tasokkaita viralliset dokumentit, perehdytysmateriaali ja esimerkkisovellukset ovat?	Kyllä

Kriteeri	Kriteerin tulkinta	Käyttö-kelpoisuus
Kirjojen lukumäärä	Kuinka monta kirjaa on julkaistu teknologiasta?	Kyllä
REST-tuki	Tukeeko teknologia REST-teknologiaa suoraan, lisäosin vai ei lainkaan.	Kyllä
Mobiilituki	Tukeeko teknologia mobiililaitteita, esim. iPhonea?	Ei
Riski	Kuinka riskialtis teknologia on ja kuinka todennäköisesti sille löytyy tukea vielä vuosien päästä? Esim. löytyykö teknologian takaa tarpeeksi laaja kehittäjäjoukko, mikä on käytettävä lisenssi, onko teknologia standardi.	Kyllä

Hyväksytty kriteerilista painoarvoineen ja pisteytyksineen

Kriteeri	Kuvaus	Paino-arvo	Pisteytys
Kehittäjien saatavuus	Kuinka paljon osaavia kehittäjiä on saatavilla kyseiselle teknologialle. Lähteinä käytetään LinkedIniä ja Monsteria.	1	Suosituin saa 10 pistettä, muut suhteutettuna suosituimpaan.
Projektin tila	Kuinka aktiivinen projekti on? Julkaisujen ja sähköpostiryhmien viestien lukumäärä kuluvana vuonna (2011).	1	Puolet pisteistä julkaisujen ja toinen puolikas sähköpostiviestien lukumäärän mukaan. Aktiivisin saa 5 pistettä / kategoria, vähiten aktiivisin saa 0 pistettä ja muut suhteutettuna aktiivisimpaan ja vähiten aktiivisimpaan.
Riski	Kuinka uusi teknologia on? Kuka ylläpitää teknologiaa? Kuinka laaja on kehittäjäyhteisö teknologian takana? Onko kyseessä standardi?	1	Puolet pisteistä jaetaan teknologian kypsyyden mukaan. Mikäli projekti on edelleen aktiivinen ja se on kestänyt vähintään 3 vuotta, saa se 5 pistettä. Mikäli projekti on tuore, mutta jo laajalti käytössä, saa se 2,5 pistettä. Muuten ei jaeta pisteitä. Toinen puolikas pisteistä jaetaan projektin aseman mukaan. Mikäli kyseessä on standardi, saa se 5 pistettä. Mikäli kyseessä on tunnetun, ison avoimenlähdekoodinyhteisön tuote, esim. Apache, saa se 2,5 pistettä. Muuten ei jaeta pisteitä.
Dokumentaation taso	Kuinka tasokkaita viralliset dokumentit, perehdytysmateriaali ja esimerkkisovellukset ovat?	1	
Oppimiskynnys	Kuinka nopeasti kehittäjä oppii käyttämään teknologiaa?	1	Kuinka monia teknologioita, ohjelmointikieliä ja sovelluskehyksiä kehittäjän on opeteltava, ennen kuin hän pystyy työskentelemään vertailtavalla teknologialla.
Lokalisaatiotuki	Tukeeko teknologia lokalisaatiota (i18n, l10n) kuinka hyvin? Kuinka helppoa lokalisoinnin käyttäminen on?	0,25	Tukee ja helppokäyttöinen = 10 pistettä, tukee ja vaikeakäyttöinen = 5 pistettä, ei tue = 0 pistettä.

Kriteeri	Kuvaus	Paino-arvo	Pisteytys
Asiakaspään validaatiotuki	Tukeeko teknologia asiakaspään validaatiota natiivisti tai sovelluslaajennoksilla? Kuinka helppoa validaation käyttö on?	0,5	Tukee ja helppokäyttöinen = 10 pistettä, tukee ja vaikeakäyttöinen = 5 pistettä, ei tue = 0 pistettä.
Työpaikka-trendit	Mikä on työpaikkojen tarjonta kullekin teknologialle ja mikä on trendi.	0,5	Suosituin saa 10 pistettä, muut suhteutettuna suosituimpaan.
Kirjojen lukumäärä	Kuinka monta kirjaa on julkaistu teknologiasta? Tutkitaan Amazon.comista.	0,75	Suosituin saa 10 pistettä, muut suhteutettuna suosituimpaan.
Mallituki	Tukeeko teknologia mallien käyttöä käyttöliittymän sivujen rakentamisessa?	1	Mikäli teknologia tukee useita mallitiedostoja, esim. PDF ja excel, on sen mahdollista saada 10 pistettä. Mikäli teknologia ei tue malleja lainkaan, ei se saa pisteitä lainkaan. Pisteet jaetaan mallitiedostojen käytön helppouden ja laadun mukaan, edellä mainittujen reunaehtojen mukaisesti.
Komponenttituki	Tukeeko teknologia komponenttien käyttöä käyttöliittymän sivujen rakentamisessa?	0,5	Mikäli teknologia tukee komponentteja ja niiden käyttäminen ja tuottaminen on yksinkertaista, saa se 10 pistettä. Mikäli teknologia ei tue komponentteja lainkaan, ei se saa pisteitä lainkaan. Pisteet jaetaan komponenttien käytön helppouden mukaan, edellä mainittujen reunaehtojen mukaisesti.
Profilointituki	Onko teknologialla mahdollista tuottaa profilointituki ja kuinka hankalaa sen käyttö on kehittäjille? Profilointituella tarkoitetaan käyttöliittymän profilointia, esim. tietojen piilottamista tai osittaista näyttämistä tähdillä korvattuna.	1	Mikäli teknologia tarjoaa natiivin mekanismin profilointituen lisäämiselle, saa se 10 pistettä. Mikäli teknologia tarjoaa tuen räätälöintien avulla, saa se 5 pistettä. Mikäli teknologia ei mahdollista profilointituen lisäämistä lainkaan, ei se saa pisteitä. Profilointituen tuottamiseen tarvittava aika ei ole niin oleellinen, koska sen tuottavat sovelluskehyksen kehittäjät kertaalleen.

Kriteeri	Kuvaus	Paino-arvo	Pisteytys
Komponentti-rajapinnan tuki	Onko sovelluskehittäjille mahdollista tuottaa heidän työtään helpottava komponentti/malli sovellusrajapinta?	0,5	Mikäli teknologia tukee selkeän rajapinnan tuottamista ja sille automaattisen täydennyksen, saa se 10 pistettä. Mikäli teknologia tukee jonkinlaisen sovellusrajapinnan määrittelyä ja/tai ei mahdollista automaattisen täydennyksen käyttämistä, saa se 5 pistettä. Mikäli kehittäjille ei ole mahdollista tuottaa sovellusrajapintaa, ei teknologia saa pisteitä.
Ajax-tuki	Tukeeko teknologia Ajax-teknologiaa suoraan, lisäosin vai ei lainkaan.	0,25	Mikäli teknologia tukee Ajaxia natiivisti, saa se 10 pistettä. Mikäli tuki on helppoa lisätä, saa se 8,5 pistettä. Mikäli tuki on hankala lisätä, saa se 4 pistettä. Mikäli teknologia ei tue Ajaxia lainkaan, ei se saa pisteitä.
Testattavuus	Kuinka helppoa teknologialla tuotettuja käyttöliittymäsivujen yksittäisiä osasia on testata?	0,25	Mikäli teknologia tukee natiivisti käyttöliittymän ja sen osasten testaamisen, saa se 10 pistettä. Mikäli testaaminen on selkeää ja helppoa ulkopuolisilla tuotteilla, saa teknologia 7,5 pistettä. Mikäli testaaminen onnistuu ulkopuolisilla tuotteilla, mutta on työlästä tai sekavaa, saa teknologia 4 pistettä. Mikäli teknologia ei mahdollista testausta lainkaan, ei se saa pisteitä.
Skaalautuvuus	Kuinka hyvin teknologia skaalautuu ympäristön resurssien mukaan, esim. sovelluspalvelimien suorituskyvyn, muistikapasiteetin ja lukumäärän mukaan. Estääkö esim. teknologian oma session käyttö skaalautumisen.	0,75	Mikäli teknologia ei estä skaalautumista, saa se 10 pistettä. Mikäli teknologia rajoittaa skaalautuvuutta hieman, saa se 5 pistettä. Mikäli teknologia rajoittaa selkeästi skaalautuvuutta, ei se saa pisteitä lainkaan.
StackOverflow	Kuinka paljon StackOverFlow kehittäjäfoorumeilla on kysymyksiä kullekin teknologialle?	0,25	Suosituin saa 10 pistettä, muut suhteutettuna suosituimpaan.
Google trends	Miltä Google Trendit näyttävät viimeisiltä 12 kuukaudelta vertailtavien teknologioiden osalta.	0,25	Suosituin saa 10 pistettä, muut suhteutettuna suosituimpaan.

Kriteeri	Kuvaus	Paino-arvo	Pisteytys
Tietoturva	Tietoturvan taso. Teknologian tulisi estää OWASP top10 listan hyökkäykset soveltuvin osin.	1	Mikäli teknologia ei estä/auta suoranaisesti estämään jotakin OWASP top10 listan käyttöliittymäkerrokseen liittyvää hyökkäystä, vähennetään teknologialta yksi piste per puuttuva ominaisuus.
Työkalutuki	Tarjoaako teknologia sitä tukevia työkaluja? Mikäli työkalutuki todella tarvitaan, mikä on työkalujen todennäköinen elinkaari ja millaisia riskejä niihin liittyy?	0,5	

Vertailumatriisin täyttämisessä käytetyt hakuarvot

Taulukko 17. Kehittäjien saatavuuden teknologiakohtaiset hakuehdot

Teknologia	Hakuehto
JSF (2.0)	JSF
Spring MVC (3.0.7)	”Spring MVC”
Wicket (1.5.2)	Wicket
GWT (2.4)	GWT
Tapestry (5.2.6)	Tapestry
Grails (1.3.7)	Grails
Vaadin (6.7.1)	Vaadin
Struts (2.2.3.1)	Struts

Taulukko 18. Työpaikkatrendien teknologiakohtaiset hakuehdot

Teknologia	Hakuehto
JSF (2.0)	java jsf
Spring MVC (3.0.7)	java ”spring mvc”
Wicket (1.5.2)	java wicket
GWT (2.4)	java gwt
Tapestry (5.2.6)	java tapestry
Grails (1.3.7)	java grails
Vaadin (6.7.1)	java vaadin
Struts (2.2.3.1)	java struts

Taulukko 19. Kirjojen lukumäärien haussa käytetyt teknologiakohtaiset hakuehdot

Teknologia	Hakuehto
JSF (2.0)	jsf 2.0
Spring MVC (3.0.7)	spring mvc 3
Wicket (1.5.2)	wicket
GWT (2.4)	gwt 2
Tapestry (5.2.6)	tapestry 5
Grails (1.3.7)	grails
Vaadin (6.7.1)	vaadin
Struts (2.2.3.1)	struts 2

Taulukko 20. StackOverflow-palvelun teknologiakohtaiset hakuehdot

Teknologia	Hakuehto
JSF (2.0)	jsf
Spring MVC (3.0.7)	spring-mvc
Wicket (1.5.2)	wicket
GWT (2.4)	gwt
Tapestry (5.2.6)	tapestry
Grails (1.3.7)	grails
Vaadin (6.7.1)	vaadin
Struts (2.2.3.1)	struts

Taulukko 21. Google Trends -palvelun teknologiakohtaiset hakuehdot

Teknologia	Hakuehto
JSF (2.0)	java jsf
Spring MVC (3.0.7)	java "spring mvc"

Teknologia	Hakuehto
Wicket (1.5.2)	java wicket
GWT (2.4)	java gwt
Tapestry (5.2.6)	java tapestry
Grails (1.3.7)	java grails
Vaadin (6.7.1)	java vaadin
Struts (2.2.3.1)	java struts

Vertailumatriisin täyttämisessä käytetyt arvosteluperusteet

Taulukko 22. Lokalisaatiotuen arvostelun perustelut

Teknologia	Perustelut
JSF (2.0)	Viestipaketit täytyy esitellä ja tuoda joka sivulle erikseen. Tämä voidaan tehdä myös mallitiedostoissa, mutta se ei sovellu kovin hyvin käyttötarkoitukseen. Toisaalta tämä on hyväkin asia, koska tällöin viestipaketit on helppo löytää ja varsinkin ylläpitäjä löytyy oikean tiedoston helposti. Kehittäjät voivat käyttää normaalia XHTML syntaksia ja *value*-attribuuttia.
Spring MVC (3.0.7)	Muuten sama kuin JSF:ssä, mutta joudutaan lisäksi käyttämään Spring:n omia tageja.
Wicket (1.5.2)	Viestipaketit ladataan automaattisesti komponentti, sivu ja sovellustasoille sekä näiden prototyypeille. Kaiken kaikkiaan erittäin helppokäyttöinen. Toisaalta tämä voi myös hankaloittaa oikean lokalisoitavan viestin löytämistä.
GWT (2.4)	Ei vaikuttanut kovin helpolta. Kehittäjä joutuu määrittelemään lokalisoitavien viestien lisäksi myös POJO:t.
Tapestry (5.2.6)	Vaikuttaa helppokäyttöiseltä ja viestipaketit ladataan automaattisesti.
Grails (1.3.7)	Sama kuin Spring MVC:ssä.
Vaadin (6.7.1)	Vaikuttaa helppokäyttöiseltä ja viestipaketit ladataan automaattisesti.
Struts (2.2.3.1)	Vaikuttaa helppokäyttöiseltä ja viestipaketit ladataan automaattisesti.

Taulukko 23. Asiakaspään validaatiotuen arvostelun perustelut

Teknologia	Perustelut
JSF (2.0)	Sisäänrakennettu tuki, mutta kaikki komponentit eivät tue Ajaxia. Ei sisällä helppoa tapaa asettaa validaatiota automaattisesti (Beans Validation ei ollut käytettävissä). Toisaalta monet komponenttikirjastot sisältävät valmiiksi Ajax-pohjaisia komponentteja.

Teknologia	Perustelut
Spring MVC (3.0.7)	Tavallaan sisäänrakennettu, mutta ei sisällä helppoa tapaa asettaa validaatiota automaattisesti
Wicket (1.5.2)	Kehittäjä toteuttaa itse JavaScriptillä.
GWT (2.4)	Esimerkkejä perustapauksesta ei löytynyt ja löytyneiden esimerkkien perusteella ei ole kaikkein helpointa.
Tapestry (5.2.6)	Sisäänrakennettu ja oikein mainiosti integroitu.
Grails (1.3.7)	Vaatii lisäosia ja kehittäjä joutuu toteuttamaan validaation JavaScriptillä itse.
Vaadin (6.7.1)	Kehittäjä joutuu laajentamaan olemassa olevia asiakas- ja palvelinpään komponentteja, mutta mekanismi on dokumentoitu kattavasti.
Struts (2.2.3.1)	Sisäänrakennettu mekanismi, joka pohjautuu erillisiin validaatio XML-tiedostoihin.

Taulukko 24. Mallitiedostojen tuen arvostelun perustelut

Teknologia	Perustelut
JSF (2.0)	Tuetut mallitiedostoteknologiat ovat Facelet (uusi standardi tapa) ja JSP (poistuva tapa). Facelet teknologia tarjoaa helppoja ja selkeitä mahdollisuuksia mallitiedostojen käyttöön. Excel ja PDF jne. tuki voidaan rakentaa suoraan palvelinpäähän itse.
Spring MVC (3.0.7)	Pääasiallinen mallitiedostojen toteutustapa on JSP. Tämän lisäksi voidaan käyttää useita muita ei-standardeja, mutta paljon käytettyjä vaihtoehtoja, kuten FreeMarker ja Velocity. Näistä standardein tapa, eli JSP, ei tarjoa läheskään yhtä hyvin mahdollisuuksia mallitiedostojen käytölle, kuin Faceletit. Excel ja PDF toiminnallisuudet eivät vaikuttaneet käyttävän juurikaan mallitiedostoja, vaan pikemminkin suoraan palvelimelle tehtävää toteutusta. Tästä annettiin kuitenkin hieman pisteitä.

Teknologia	Perustelut
Wicket (1.5.2)	Mallitiedostojen käyttö ei vaikuttanut kovin intuitiiviselta käytettävissä olleen ajan perusteella. Ominaisuus olisi voinut osoittautua pidemmässä tarkastelussa toimivaksi, mutta tutustumiseen varattiin saman verran aikaa, kuin muillakin teknologioilla. Tämän perusteella, muiden teknologioiden mallitiedostoratkaisut olivat selkeämpiä ja intuitiivisempia. Mallitiedostot käyttävät HTML syntaksia. Ei sisällä suoraa Excel, eikä PDF tukea ja näiden lisääminen vaikuttaa työläämmältä, kuin esim. JSF ja Spring tapauksissa.
GWT (2.4)	Ei tarjoa vakuuttavaa mallitiedostoratkaisua, mutta kehittäjät voivat käyttää valitsemaansa mallitiedostoteknologiaa itse. Excel ja PDF jne. tuki voidaan rakentaa suoraan palvelinpäähän itse.
Tapestry (5.2.6)	Mallitiedostojen käyttö vaikuttaa selkeältä. Excel ja PDF jne. tuki voidaan rakentaa suoraan palvelinpäähän itse.
Grails (1.3.7)	Mallitiedostojen käyttö ei vaikuttanut kovin intuitiiviselta käytettävissä olleen ajan perusteella. Ominaisuus olisi voinut osoittautua pidemmässä tarkastelussa toimivaksi, mutta tutustumiseen varattiin saman verran aikaa, kuin muillakin teknologioilla. Tämän perusteella, muiden teknologioiden mallitiedostoratkaisut olivat selkeämpiä ja intuitiivisempia. Pääasiassa mallitiedostot toteutetaan Groovy syntaksilla. Ei sisällä suoraa Excel, eikä PDF tukea ja näiden lisääminen vaikuttaa työläämmältä, kuin esim. JSF ja Spring tapauksissa.
Vaadin (6.7.1)	Vaikuttaa siltä, ettei tue lainkaan mallitiedostojen käyttöä.
Struts (2.2.3.1)	Pääasiallinen mallitiedostojen toteutustapa on JSP. Tämän lisäksi voidaan käyttää useita muita ei-standardeja, mutta paljon käytettyjä vaihtoehtoja, kuten FreeMarker ja Velocity. Näistä standardein tapa, eli JSP, ei tarjoa läheskään yhtä hyvin mahdollisuuksia mallitiedostojen käytölle, kuin Faceletit. Excel ja PDF toiminnallisuudet eivät vaikuttaneet käyttävän juurikaan mallitiedostoja, vaan pikemminkin suoraan palvelimelle tehtävää toteutusta. Tästä annettiin kuitenkin hieman pisteitä.

Taulukko 25. Komponenttituen arvostelun perustelut

Teknologia	Perustelut
JSF (2.0)	Komponenttipohjainen. Versiosta 2.0 lähtien omien komponenttien tuottaminen on helppoa ja niitä voidaan tehdä useilla tavoilla.
Spring MVC (3.0.7)	Ei vaikuta tukevan komponentteja.
Wicket (1.5.2)	Melko helppo tuottaa komponentteja. Komponentit ovat Java-luokkia.
GWT (2.4)	Melko helppoa tuottaa komponentteja (Widget).
Tapestry (5.2.6)	Vaikuttaa melko helpolta ja intuitiiviselta tuottaa komponentteja.
Grails (1.3.7)	Ei vaikuta tukevan komponentteja.
Vaadin (6.7.1)	Todella intuitiivinen ja selkeä komponenttituki.
Struts (2.2.3.1)	Jonkinlainen tuki komponenteille JSP-tageilla.

Tietojenvälitystapojen vertailu

URL-parametrit

Hyvää vaihtoehdossa on

- PRG (Post/redirect/get) suunnittelumallin mukainen
- toimivat kirjanmerkit ja sähköpostilinkit
- erilliset selainikkunat ja -välilehdet mahdollisia
- selailuhistoria eteen/taakse ja ikkunan virkistys toimivat
- lomakkeiden kaksinkertaisen lähetyksen esto
- käsiteltävä ja näytettävä tieto on ”tuoretta”
- pakottaa ottamaan kantaa käyttäjän lähettämän tiedon tarkistamiseen.

Huolenaiheena vaihtoehdossa on liiketoimintalogiikan oikominen, mikäli käyttäjä tietää mitä on tekemässä ja sovelluksessa ei ole mietitty käyttäjän tietojen käsittelyä.

Vaihtoehtoon liittyy myytti, että muissa vaihtoehdoissa käyttäjän lähettämää dataa ei tarvitse tarkistaa ja puutteellisiin tietoihin puuttua. Nämä koskevat kuitenkin kaikkia vaihtoehtoja. Liiketoimintalogiikassa ei saa koskaan luottaa käyttäjän lähettämään dataan.

Sessiotietokanta + URL-parametri

Hyvää vaihtoehdossa on

- käyttäjä näkee URL:ssa vain sessioavaimen
- sessiot säilyvät haluttaessa päiviä.

Huonoa vaihtoehdossa on

- kuinka hyvin toimii useiden ikkunoiden ja välilehtien kanssa?
- kirjanmerkit ja linkit eivät toiminnassa
- sessiot säilyvät päiviä, mitä käy kun käyttäjällä useampia sessioita eri konteksteilla käytössä?
- sessioon tallennetut tiedot saattavat olla vanhentunutta
- tuotantoympäristön lisääntynyt kompleksisuus

- session yhtäaikainen päivitys eri sovellusten toimesta
- uusi alustakomponentti ylläpidettäväksi.

Vaihtoehtoon liittyy myytti, että se heikentäisi suorituskykyä dramaattisesti. Tiedot täytyy joka tapauksessa palauttaa sessioon jostain, eikä hyvin suunniteltu tietokanta aiheuta huomattavaa eroa suorituskyvyssä.

HTTP-otsikkotiedot

Hyvää vaihtoehdossa on, että tiedot pysyvät piilossa käyttäjältä.

Huonoa vaihtoehdossa on

- toimiiko muuten kuin teoriatasolla
- uudelleenohjaus hajottaa toiminnallisuuden
- siteMinder jne. filtterit saattavat rikkoa toiminnallisuutta
- saattaa aiheuttaa todella hankalasti selvitettäviä tuotanto-ongelmia
- selaimissa saattaa olla eroja otsikkotietojen käsittelyssä
- kirjanmerkit ja linkit eivät toiminnassa.

Session-/tiedonjako erillisellä kirjastolla

Hyvää vaihtoehdossa on

- käyttäjä näkee URL:ssa vain sessioavaimen
- sessiot säilyvät tarvittaessa päiviä.

Huonoa vaihtoehdossa on

- kuinka hyvin toimii useiden ikkunoiden ja välilehtien kanssa?
- kirjanmerkit ja linkit eivät toiminnassa
- sessiot säilyvät päiviä, mitä käy kun käyttäjällä useampia sessioita eri konteksteilla käytössä?
- sessioon tallennetut tiedot saattavat olla vanhentunutta
- tuotantoympäristön lisääntynyt kompleksisuus
- session yhtäaikainen päivitys eri sovellusten toimesta
- uusi tuote, uudet lisenssikustannukset ja tuotanto-ongelmat.

Vaihtoehtoon liittyy myytti, että se heikentäisi suorituskykyä dramaattisesti. Tiedot täytyy joka tapauksessa palauttaa sessioon jostain, eikä hyvin suunniteltu toteutus aiheuta huomattavaa eroa suorituskyvyssä.

Portaali

Hyvää vaihtoehdossa on

- tiedon välitys mahdollista PortletContextin kautta
- jos otettaisiin huomioon sovelluksia suunniteltaessa, saatettaisiin haluttaessa tarjota tulevaisuudessa portlet-sovelluksia esim. REST-pohjaisesti suoraan asiakkaan omille asiakkaille.

Huonoa vaihtoehdossa on

- kallis ja raskas ratkaisu tarpeisiin nähden
- portaalille ei muuta tarvetta
- monesti melko komplekseja tuotteita
- sovellukset joudutaan asentamaan samaan ajoympäristöön
- ympäristöstä ei kuitenkaan otettaisi hyötyä irti.